Angerhausen · Becker · Gerits
Schellenberger



64 für Profis

Anwendungsprogrammierung in BASIC für Fortgeschrittene

EIN DATA BECKER BUCH

ISBN-Nr.: 3-89011-007-X

Wichtiger Hinweis

VORWORT

Mit "64 für Profis" können wir Ihnen bereits den siebten Titel in der beliebten Reihe der DATA BECKER BÜCHER vorlegen. Das Erscheinungsdatum dieses Buches fällt zusammen mit einer Zahl, auf die wir besonders stolz sind:
In nur 10 Monaten konnten wir über 100.000 DATA BECKER Bücher verkaufen.

Auch bei "64 für Profis" hat sich unser Autorenteam wieder größte Mühe gegeben, Ihnenn fundierte Ratschläge und Anwendungen zu geben, die Ihnen helfen, Ihren Commodore 64 optimal einzusetzen.

Anwendungsprogrammierung in BASIC für Fortgeschrittene ist das Thema dieses Buches. Vom Programmentwurf über Menuesteuerung, Bildschirmmaskenaufbau, Parametrisierung, Datenzugriff und Druckerausgabe bis hin zur Dokumentation wollen wir Ihnen mit vielen Beispielen zeigen, wie gute Anwendungsprogrammierung in BASIC vor sich geht.
Besonders stolz sind wir dabei auf die von Klaus Gerits entwickelte neue Dateizugriffsmethode QUISAM.
Neben den auf dem Titel vermerkten Autoren, die Ihnen alle bereits von anderen DATA BECKER BÜCHERN bekannt sein dürften, halfen bei der Erstellung dieses Buches die Herren Englisch, Froitzheim, Kausmann und Retzlaff. Die Zeichnungen stammen von Frau Jordan. Ihnen allen gilt unser besonderer Dank.

Vorwort zur 2. überarbeiteten und erweiterten Auflage

Innerhalb von nur drei Monaten war die erste Auflage dieses Buches mit 20.000 Exemplaren vergriffen. Wir haben die Gelegenheit der Neuauflage genutzt, ein paar interessante Kapitel hinzuzufügen.

Viel Spaß bei der Lektüre dieses Buches wünschen

Ihre Autoren

Inhaltsverzeichnis

1. Kapitel:

2. Kapitel:

3. Kapitel:

4. Kapitel:

5. Kapitel:

EINLEITUNG - PROGRAMMIEREN PROFIS ANDERS ?

Ja und Nein. Profis programmieren vor allem für einen anderen Zweck. In der Regel erstellen sie Programme, die nicht von ihnen selbst, sondern von anderen Leuten benutzt werden. Daraus ergibt sich zwangsläufig eine andere Methodik bei Planung und Durchführung der Programmerstellung.

Denken Sie doch einmal zurück an Ihre eigene Anfänge in der Programmierung. Da kam es Ihnen zunächst darauf an, überhaupt etwas mit Ihrem COMMODORE 64 zu erreichen. Jede Reaktion des Bildschrims, immer, wenn sich etwas tat, erfüllte Sie mit Stolz. Mit wachsender Kenntniss neuer Befehle und Programmiertricks wuchsen auch Ihre Programme. Oft fanden Sie sich dann selbst nicht mehr zurecht und die Fehlersuche wurde manchmal zur Qual. Stellen Sie sich einmal vor, mit diesem Programm hätte jemand anderes arbeiten müssen !

Ganz anders beim Programmierprofi. Für ihn ist Programmieren weder Selbstzweck noch Hobby (was nicht ausschließt, daß es ihm trotzdem Spaß macht), sondern die Erledigung eines bestimmtem Programmierauftrages. Diesen Programmierauftrag gilt es, so gut und schnell wie irgend möglich zu erfüllen. Dabei erweisen sich drei Dinge als besonders wichtig:

* effiziente Programmerstellung
  Zeit kostet Geld. Deshalb muß der Programmierprofi sein Ziel, nämlich die Fertigstellung eines lauffähigen Programms, möglichst schnell mit vertretbarem Aufwand erreichen. Ein neues Programm ist für ihn kein Geniestreich, sondern das Ergebnis sorgfältiger Planung und Vorbereitung. Dabei macht sich der Programmierprofi etwas

zunutze, was man in Anlehnung an die Textverarbeitung auch als Bausteinverarbeitung bezeichnen könnte. Für ständig wiederkehrende Dinge - bei ähnlichen Anwendungen sind oft 80 % und mehr gleich - schafft er sich Module (Maskenaufbau, Dateiverwaltung, Datumsroutinen, Rechenformeln etc.), aus denen er schnell das Grundgerüst einer neuen Anwendung zurechtzimmern kann. So kann er das Hauptaugenmerk von der Codierung hin zur Organisation lenken. Sorgfältige Planung des Programms schützt den Profi vor Sackgassen und Umwegen in der Programmierung. Wer sein Programm vorher richtig plant, der ist hinterher auf das bei Hobbyisten so beliebte RENUMBER nicht mehr angewiesen. Nicht so sehr die genialen Programmiertricks sind es also, die den Programmierprofi auszeichnen (hier stehlen ihm oft engagierte Hobbyisten die Schau), sondern die zielgerechte Arbeit.

* der Gedanke an den Benutzer

  Wer Programme erstellt, die nicht von ihm selbst benutzt werden, muß entsprechend sorgfältig vorgehen. Solche Programme dürfen nicht unvermutet aussteigen oder aber ohne jede Bedienerführung am Bildschirm den Benutzer zum ratlosen Dauerleser des (hoffentlich!) vorhandenen Handbuches machen.

  Hierzu ein Beispiel: Denken Sie einmal an eine bestimmte Strecke, die Sie häufig mit dem Auto fahren. Für Sie selbst kein Problem, denn Sie wissen exakt wo's langgeht und fahren eine solche Strecke praktisch im Schlaf. Jetzt sorgen Sie einmal dafür, daß ein Ortsunkundiger diese Strecke ebenfalls mühelos zurücklegen kann, ohne sich zu verfahren, und zum Ziel gelangt. Zwei Hilfsmittel haben Sie hier zur Verfügung. Das eine ist die Beschilderung, die der Bedienerführung am Bildschrim entspricht und auf die Sie gottseidank mehr Einfluß haben als auf die Beschilderung

des Straßenverkehrs. Das andere ist eine schriftliche Wegbeschreibung, entsprechend der Programmbeschreibung bzw. des Bedienungshandbuches. Erkennen Sie jetzt den Unterschied zwischen eigenen Programmen für den Hausgebrauch und solchen Programmen, die für fremde Benutzer geschrieben sind ? Wer Programme erstellen will, die professionellen Ansprüchen genügen sollen, muß dies so tun, daß sich auch "Ortsunkundige" zurechtfinden.

* Änderungsfreundlichkeit
  Programme werden für gewisse Umweltbedingungen erstellt, und die können sich leicht ändern. Sehr wichtig ist deshalb, daß sich Programme auch leicht ändern und anpassen lassen können. Gravierende Denkfehler bei der Programmerstellung, neue, unvorhergesehene Wünsche des Programmbenutzers oder aber veränderte Rahmenbedingungen, wie z.B. ein neuer Mehrwertsteuersatz, eine neue Lohnsteuerformel oder eine neue Firmenbezeichnung sind für ein professionell erstelltes Programm kein Problem. Wehe aber, wenn so ein Fall eintritt, und Sie müssen sich durch 20 K Spaghetticode durcharbeiten, wo jede Fehlerbeseitigung neue Fehler provoziert, und wo Sie sich sehnsüchtig einen CURSOR wünschen, der auch als Blindenhund funktioniert.

Ziel dieses Buches ist nicht, aus Ihnen einen Superprogrammierer zu machen, der ein neues VISICALC oder einen so großen Wurf wie das bekannte WORDSTAR aus dem Ärmel zaubert. Wir möchten Ihnen lediglich zeigen, wie Sie mit weniger Aufwand mehr erreichen und Programme erstellen, an denen nicht nur Sie selbst sondern auch andere Leute ungetrübte Freude haben können. Ihr Computer, der COMMODORE 64, bietet dafür hervorragende Voraussetzungen. Mit 64 K Speicher bietet er mehr, als vor ein paar Jahren noch fortschritliche Anlagen der Mittleren Datentechnik vorweisen

konnten. Durch die wichtigen Funktionstasten kommt die Tastatur schon der professionellen Bildschirmgeräte nahe. Mit theoretischem Ballast wollen wir Sie in diesem Buch möglichst wenig langweilen, auch wenn dies manchmal unumgänglich ist. Stattdessen wollen wir Ihnen Ratschläge und erprobte Rezepte geben, die Sie direkt in die Tat umsetzen können.

Gute Manieren und zielgerechtes, planvolles Vorgehen lohnen sich, auch und gerade in der Programmierung!

KAPITEL 1: BEVOR SIE ANFANGEN ZU PROGRAMMIEREN

1.1. Das sollten Sie sich ersparen !

Die meisten Hobby-Programmierer (gelegentlich auch sogenannte Profi's) entwickeln ihre Programme ohne jede Vorarbeit direkt am Bildschirm. Das heißt: Von der Programmidee bis zum fertigen Programm wird alles ausschließlich an der Maschine erledigt. Bei einfachen Programmen ist dies sicherlich noch möglich, bei größeren und anspruchsvolleren Programmen aber nicht mehr.

Sicherlich ist es Ihnen auch schon so gegangen: Sie hatten plötzlich eine Idee zu einem neuen Programm. Da haben Sie sich dann einfach vor Ihren COMMODORE 64 gesetzt, die Maschine eingeschaltet und anschließend sofort drauflos programmiert. Zu Anfang ging das ganz gut. Schon nach wenigen Programmzeilen, die auch noch gut zu überblicken waren, tat sich bereits etwas am Bildschirm. Aber mit dem größer werdenden Programm und der sich bei der Programmierung entwickelnden Idee (!) kamen dann plötzlich die Probleme. Mit der Präzisierung Ihrer ursprünglichen Programmidee wurde Ihr Programm immer unübersichtlicher, komplizierter und verschachtelter. Aber da gab es dann ja noch so etwas wie einen Programmer's Toolkit. Mit FIND, DUMP, RENUMBER und anderen Hilfsbefehlen wurde dann herumgedoktert und an den Symptomen laboriert, nicht aber an der Ursache! Auch das half schließlich nichts mehr. Ihr Durchblick schwand, die Fehler häuften sich, und aus Lust wurde Frust. Selbst wenn das Programm dann irgendwann doch noch fertig wurde - ganz fertig werden solche Programme in der Regel nie - so hinterließ es bei Ihnen doch wenig Befriedigung.

Was war passiert? Sie hatten in guter alter Hackermanier

Codes in Ihren Commodore 64 'gehackt', aber nicht programmiert. Zum Programmieren gehört nämlich eine ganze Menge mehr. Das, was Sie gemacht haben, nennt man Codieren. Man versteht hierunter das Umsetzen der einzelnen Programmschritte in eine für den Computer verständliche Form, z.B. BASIC. Zum Programmieren gehört aber eine ganze Menge mehr als nur das Codieren. Es beginnt mit der sorgfältigen Ausarbeitung der Programmidee. In der Fachsprache nennt man das Programm - bzw. Systemanalyse. In der Groß-EDV werden speziell für diese Aufgabe hochbezahlte und entsprechend qualifizierte Systemanalytiker beschäftigt. Deren Aufgabe besteht einzig und allein darin, mögliche neue Anwendungen und Programmideen so exakt zu analysieren und aufzubereiten, daß diese anschließend problemlos in Codes umgesetzt werden können.

Da Sie sich mit Sicherheit keinen eigenen Systemanalytiker leisten können, zeigen wir Ihnen in diesem Kapitel, wie eine sorgfältige Programmanalyse vor sich gehen sollte. Dabei erfahren Sie keine genialen Programmiertrick's, sondern bekommen lediglich gezeigt, wie man mit sorgfältigem, schrittweisem Vorgehen eine Menge Pannen und lästige Programmierhindernisse umgehen kann.

1.2 Die Programmidee

Fast jeder Computerfreak ist ständig auf der Suche nach neuen Ideen, die er programmtechnisch auswerten kann. Auch Sie, lieber Leser, können sicherlich ein Liedchen davon singen. Die meisten Programmierer setzen sich - nachdem ihnen ein geeigneter Einfall gekommen ist - sofort an ihren Computer und versuchen direkt, das Programm zu schreiben. Erfahrungsgemäß muß hier ein weiter Weg begangen werden,der mit allen möglichen Hindernissen gepflastert ist. Genau diese programmtechnischen Hindernisse wollen wir versuchen auszuschalten, bevor der Computer überhaupt eingeschaltet wird. Grundlegend hierfür ist eine detaillierte Ausarbeitung des betreffenden Problems von Anfang an.

Am Anfang eines jeden Programmes steht immer die Idee. Da man im Moment des Einfalls meistens am besten weiß wie das zukünftige Programm aussehen soll ist es am besten, wenn man die Programmidee in kurzen Stichworten umschreibt. Später (am Schreibtisch) sollten Sie genau festlegen was Ihr Programm einmal können soll. Tragen Sie jetzt alle nötigen Problemstellungen und sonstige Einzelheiten für Ihr späteres Programm zusammen. Vielleicht haben Sie schon einmal Programme entwickelt, die in der Problemstellung ähnlich gelagert waren. Sicherlich werden Sie in diesen Programmen Hilfsroutinen finden, die Sie komplett in das neue Projekt einbauen können.

Wenn Sie Ihre Programmidee sorgfältig beschrieben haben, ist der erste Schritt in der Programmvorbereitung getan. Dieser erste Schritt wird in der Fachwelt als PROBLEM-ANALYSE bezeichnet. Betrachten Sie diese Problemanalyse einfach als eine Art Stoffsammlung. An Hand von kleinen Beispielen wollen wir Ihnen erläutern, wie eine solche Problemanalyse am zweckmäßigsten zu Erstellen ist. Hierbei ist es noch relativ unbedeutend, wie das Problem später in eine,

dem Computer verständliche Sprache umzusetzen ist.

Beispiel 1: Problemanalyse für eine Adressenverwaltung

Wir wollen jetzt einmal darangehen, das Problem einer kleinen Adressenverwaltung zu analysieren. Hierzu stellen wir uns gleich zu Beginn die Frage, die mit der Erstellung einer Problemanalyse im engen Zusammenhang steht:

Was will ich eigentlich programmieren ?

Unser Programm soll es uns ermöglichen, Adressen einzugeben, diese auf ein externes Speichermedium abzuspeichern und wieder einzulesen. Ferner soll eine kleine Suchroutine vorgesehen werden, über die einzelne Daten gesucht und auf dem Bildschirm ausgegeben werden können.

Nachdem nun definiert ist was das Programm können muß, wollen wir unser Problem (chen) in einzelne Teilprobleme zerlegen. Jedes dieser Teilprobleme wird einzeln analysiert und unter Berücksichtigung aller nötigen Kriterien sorgfältig geplant. Unsere Adressenverwaltung gliedert sich in die folgenden Teilprobleme:

A) DATENEINGABE
B) DATEI VERWALTEN
C) DATEN SUCHEN

A) Dateneingabe

Hier müssen wir uns vor allem darüber Gedanken machen, welche Daten eigentlich gespeichert werden sollen. Legen Sie die einzelnen Kriterien fest, die gespeichert werden sollen und bestimmen sie, wieviele Zeichen jedes Kriterium maximal enthalten darf. Aus der Datensatzlänge ergibt sich später die maximale Anzahl der Datensätze, die in dem Arbeitsspei-

cher Ihres Commodore 64 gespeichert werden können. Ein kleines Rechenbeispiel soll dies veranschaulichen. Für unsere Adressenverwaltung sollen die Eingabekriterien wie folgt festgelegt werden:

```
           Nachname  => 15  Zeichen
           Vorname   => 15  Zeichen
           Strasse   => 15  Zeichen
           PLZ       =>  4  Zeichen
           Telefon   => 13  Zeichen
           Bemerkung => 25  Zeichen
           ========================
      Datensatzlänge => 87  Zeichen
```

Wie Sie wissen, darf bei Ihrem Commodore 64 der Speicherplatz (in BASIC) von 38911 Bytes nicht überschritten werden. Beachten Sie, daß unser fertiges Programm auch noch einen Teil dieses Speicherplatzes in Anspruch nehmen wird. Da wir jetzt noch nicht wissen wie lang unsere Adressenverwaltung werden wird, kann hier noch keine Aussage über die maximal mögliche Anzahl von Datensätzen gemacht werden. Da Sie jedoch sicherlich wissen, daß ein Byte stellvertretend für jeweils ein Zeichen steht, können Sie später über die FRE(X)-Funktion leicht den verbleibenden Speicherplatz ermitteln und diese Zahl durch die Datensatzlänge dividieren. In unserem Fall wäre dies eine Datensatzlänge von 87 Zeichen. Nehmen wir einmal an, es stünde uns uns noch ein freier Arbeitsspeicher von 29568 Bytes zur Verfügung:

$$29568 : 87 = 339.86$$

Bei diesem angenommenen Wert wäre es möglich, 339 Adressen zu verwalten. Für unser Programm wollen wir einmal 'tiefstapeln' und eine maximale Datensatzzahl von 200 festlegen. Nun haben wir alles Wissenswerte über die Dateneingabe niedergeschrieben. Mehr gehört hier nicht rein ! Wenden wir uns nun dem nächsten Problem, der Dateiverwaltung, zu.

B) Datei verwalten

Da wir unsere Daten nicht jedesmal wieder neu eingeben wollen, muß eine Möglichkeit zur Datenspeicherung auf externe Datenträger vorgesehen werden:

Wir brauchen eine Datei!

Hier müssen wir uns gleich die Frage nach der Dateiart stellen: Sequentielle - oder Direktzugriffsdatei ?

Auf Ihrem Commodore 64 ist es sicherlich am einfachsten, eine sequentielle Datei aufzubauen (mit einer DATASETTE sind sogar nur sequentielle Dateien möglich). Wenn Sie ein Floppy 1541 besitzen, haben Sie die Wahl zwischen drei möglichen Dateiarten :

Sequentielle Dateien
Direktzugriffsdateien
Relative Dateien

Wir wollen unser Programm mit einer sequentiellen Datei aufbauen, um den vielen Anwendern von Datasetten Rechnung zu tragen. Sollten Sie stolzer Besitzer einer Floppy 1541 sein wird Ihnen zum Thema Dateiverwaltung unser 'Großes Floppy Buch' sicherlich wertvolle Hinweise geben können, die in Ihrem Bedienungshandbuch nicht gegeben werden.

Die Datei wird sequentiell organisiert !

Sollen mit einem einzigen Adressenverwaltungsprogramm mehrere Dateien verwaltet werden (Bekannte, Firmenanschriften, Hoteladressen u.s.w.), so muß ein weiterer Faktor berücksichtigt werden:

Der Dateiname muß variabel sein !

Besonders bei der Dateiverwaltung mit der Floppy 1541 ist dieser Faktor sehr bedeutend. Für jede gespeicherte Datei muß ein eigener Name vergeben werden, da das DOS dieser Diskettenstation anhand dieser Bezeichnungen die einzelnen Dateien auf der Diskette verwaltet. Wenn Sie eine Datasette zur Datenspeicherung verwenden wollen ist es durchaus möglich, mehrere Dateien mit dem selben Namen abzuspeichern. Dies ist jedoch nicht empfehlenswert, da dadurch Ihre Datensammlung mit der Zeit relativ unübersichtlich wird. Vergeben Sie deshalb bei jeder neuen Datei einen neuen,geeigneten Namen (unter geeignet verstehen wir den Zusammenhang zwischen Dateinamen und dem Inhalt der gespeicherten Datei).

Je nach Komfort des Programmes müssen Sie nun weitere Überlegungen im Bezug auf die Dateiverwaltung anstellen:

- Soll die Datei erweiterbar sein ?
- Müssen die gespeicherten Daten von Zeit zu Zeit geändert werden (z.B. Lagerverwaltung) ?
- Sollen komplette Dateien vom Programm aus gelöscht werden können ?
- Kann eine,bereits gespeicherte Datei von einer neuen Datei überschrieben werden ?

Für unsere Adressverwaltung wollen wir von den obenstehenden Fragen die erste - Soll eine Datei erweiterbar sein ? - in unser Beispiel einarbeiten.

Die Datei muß erweiterbar sein !

Diese Anforderung ist gerade bei einer Adressenverwaltung von großer Bedeutung. Lassen Sie Ihre Datei mit dem immer größer werdenden Bekanntenkreis wachsen. Bei einer sequentiell organisierten Datei geht dies folgendermaßen vor sich:

1. Datei komplett einlesen.
2. Datei über den Eingabemodus erweitern.
3. Datei komplett abspeichern.

Bei diesen sequentiellen Dateien sind Sie dazu gezwungen, eine weitere Forderung an Ihr Programm zu stellen:

Die geänderte Datei muß die alte überschreiben !

Nun sind alle Anforderungen an unsere Adressdatei gestellt. Wir wollen uns nun Gedanken darüber machen,wie die Datei auf dem externen Speichermedium (Cassette oder Diskette) angelegt werden soll. Das heißt, wir müssen festlegen welche Informationen in welcher Reihenfolge abgespeichert werden:

1. Dateiname
2. Anzahl der Datensätze
3. Datensätze

Die Punkte 1 und 2 sind vor allem dann nötig, wenn die betreffende Datei geändert werden soll. Die Anzahl der gespeicherten Datensätze kann im Programm für eine Schleifensteuerung, z. B. in der Suchroutine sehr nützlich sein.

Wir haben jetzt alle Forderungen und den Aufbau unserer Datei festgelegt und beginnen mit der Analyse des dritten Problemes.

C) Daten suchen

Zuerst müssen Sie sich darüber klar werden, wie und nach welchen Kriterien die einzelnen Informationen gesucht werden sollen. Genügt es, eine einzige Suchroutine zu schreiben, die nach Eingabe eines Suchkriteriums (in unserem Beispiel könnten das Name, Vorname,Postleitzahl etc. sein) nach dem entsprechenden Datensatz sucht oder ist es sinnvoll ein weiteres Suchsystem in das Programm mit einzubauen ?

Stellen Sie sich vor, Sie wollen sich mit Ihrem Programm die

Adresse eines Bekannten ausgeben lassen und wissen nicht mehr genau, wie der Name des betreffenden geschrieben wird. Hier kann es vorkommen, daß Sie immer wieder eine Fehlermeldung erhalten, da der Computer genau nach dem sucht was ihm auch eingegeben wurde (ein Ähnlichkeitsvergleich wird nicht durchgeführt). Hier ist es zweckmäßig, eine zweite Zugriffsmöglichkeit auf die gespeicherten Daten zu schaffen:

Auflisten nach einem Schlüsselfeld !

Legen Sie fest, welches Ihrer Datenelemente das Schlüsselfeld sein soll, das stellvertretend für den jeweiligen Datensatz in der Bildschirmliste ausgegeben werden soll. In unserem Beispiel soll der Nachname als Schlüsselfeld deklariert werden. Mit Hilfe dieser Schlüsselfelder können Sie nun jederzeit überprüfen, ob bei der Eingabe ein Schreibfehler vorlag, indem Sie sich die Liste auf dem Bildschirm ausgeben lassen und visuell den betreffenden Datensatz suchen.

Zum Zweiten soll eine Suchroutine im Programm enthalten sein, die nach Eingabe des gewünschten Suchkriteriums den entsprechenden Datensatz automatisch heraussucht. In unserem Beispielprogramm können Sie unter den folgenden Kriterien wählen:

1. Name
2. Vorname
3. Strasse
4. Wohnort
5. Telefon
6. PLZ

Hier möchten wir die Problemanalyse für unser Beispiel abschließen. Verfahren Sie bei Ihrer eigenen Programmplanung bitte nach dem gleichen 'Strickmuster' wie wir es in diesem Kapitel beschrieben haben. Mit dieser Stoffsammlung haben Sie dann bereits den ersten Grundstein für eine professionelle Programmerstellung gelegt.

1.3. Der Programmentwurf

Als Grundlage für den Programmentwurf dient Ihnen die Problemanalyse (Stoffsammlung), die wir gemeinsam in Kapitel 1.2. erstellt haben. Dort haben wir die wesentlichen Punkte, die zur Problemlösung mit Hilfe eines Computers benötigt werden, zusammengetragen. Bei großen und komplizierten Problemstellungen ist es zu empfehlen, die gesamte Problematik in kleinere Teilprobleme zu gliedern. Entwickeln Sie nun ein grobes Gerüst der geplanten Programme (Unterprogramme). Hierbei kommt es immer noch nicht darauf an, Einzelheiten zur späteren programmtechnischen Lösung zu erläutern. Zum Entwurf dieses Programmskeletts gibt es verschiedene, graphische Hilfsmittel. Das bekannteste Hilfsmittel zur logischen Darstellung eines Datenflußes ist zweifellos das Flußdiagramm nach DIN 66001. Auf diese Methode wollen wir hier näher eingehen. Im Kapitel 1.5. beschreiben wir eine weitere Möglichkeit zur graphischen Lösung von Problemen, die vorwiegend in der strukturierten Programmierung angewandt wird: Das Struktogramm nach Nasi-Schneiderman.

1.3.1. Das Flußdiagramm

Mit einem Flußdiagramm können Sie vor allem für EDV-Laien den Datenfluß eines Problemes sehr anschaulich und übersichtlich gestalten. Der einfache und unkomplizierte Aufbau solcher Ablaufdiagramme hat dafür gesorgt, daß diese Darstellungsart am weitesten verbreitet ist. In allen, branchenorientierten Massenmedien (CHIP, MC etc.) finden Sie regelmäßig Flußdiagramme, die die dort dargebotenen Programme dokumentieren. Vorraussetzung für eine sinnvolle Anwendung von Flußdiagrammen ist die Kenntnis der Zeichen und Symbole, die bei dieser Darstellungsart verwendet werden.

Im Folgenden wollen wir Ihnen diese Symbole, die nach DIN 66001 festgelegt sind, kurz erläutern:

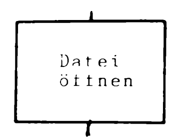
1. Grundelemente:

Als erstes wollen wir Ihnen die eigentlichen Basissymbole zeigen. Mit Kenntnis dieser wenigen Zeichen sind Sie bereits in der Lage, kleinere und unkomplizierte Flußdiagramme selbst zu entwerfen.

Dieses Symbol steht für den Programmstart bzw. für das Programmende. Auch Programmein - und Programmaussprünge werden mit diesem Zeichen gekennzeichnet:

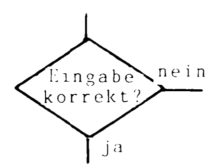


Mit dem folgenden Zeichen werden Anweisungen, also Tätigkeiten oder Vorgänge die im Computer oder vom Programm erledigt werden, dargestellt. Hierbei steht im Symbol ein Text, der auf die Art des Verarbeitungsschrittes hinweist:

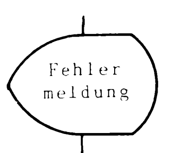


In der nun folgenden Raute werden immer Entscheidungen abverlangt, nach deren Bedingungen (ja/nein) das Programm verzweigen soll. Bei der Anwendung dieses Symboles in einem

Flußdiagramm ist es unwesentlich, ob die Antwort auf die gestellte Frage vom Computer oder durch den Anwender beantwortet wird (z.B. bei einer Abfrage, ob ein Programm wiederholt werden soll):



Als letztes Grundsymbol soll das Zeichen für die Bildschirmausgabe bzw. ein akkustisches Signal vorgestellt werden.



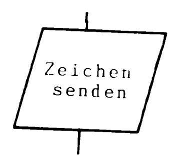
An dieser Stelle noch ein paar Worte zur Normung dieser Symbole: Alle Symbole sind nach DIN 66001 genormt. Trotzdem kann man immer wieder feststellen, daß fast jeder Programmierer seinen eigenen Stil bei der Darstellung von Flußdiagrammen entwickelt hat, mit dem er einfach besser zurecht kommt. Dies ist sicherlich nicht weiter tragisch. Wenn jedoch mehrere Programmierer an dem ein und selben Projekt arbeiten, ist eine normgerechte Anwendung unerläßlich (was würde wohl geschehen, wenn mehrere verschiedene Zeichenstile zu einem fertigen Programm zusammengefügt werden sollen ?).

2. Weitere Symbole

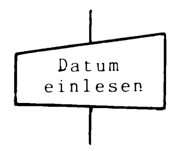
Nachfolgend werden weitere Symbole beschrieben, die am häu-

figsten bei Darstellung in Flußdiagrammen verwendet werden. Hierbei erhebt diese Liste keinen Anspruch auf vollständigkeit, da wir nur auf die wesentlichen Zeichen und Symbole eingehen wollen.

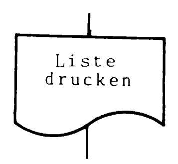
Ein-/Ausgabe (INPUT/OUTPUT):



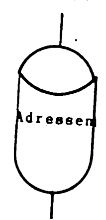
Manuelle Eingabe (MANUAL INPUT):



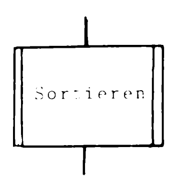
Listenausgabe (DOCUMENT):



Speicherung auf Diskette (ONLINE STORAGE):



Unterprogrammaufruf (CALL SUBPROGRAM):



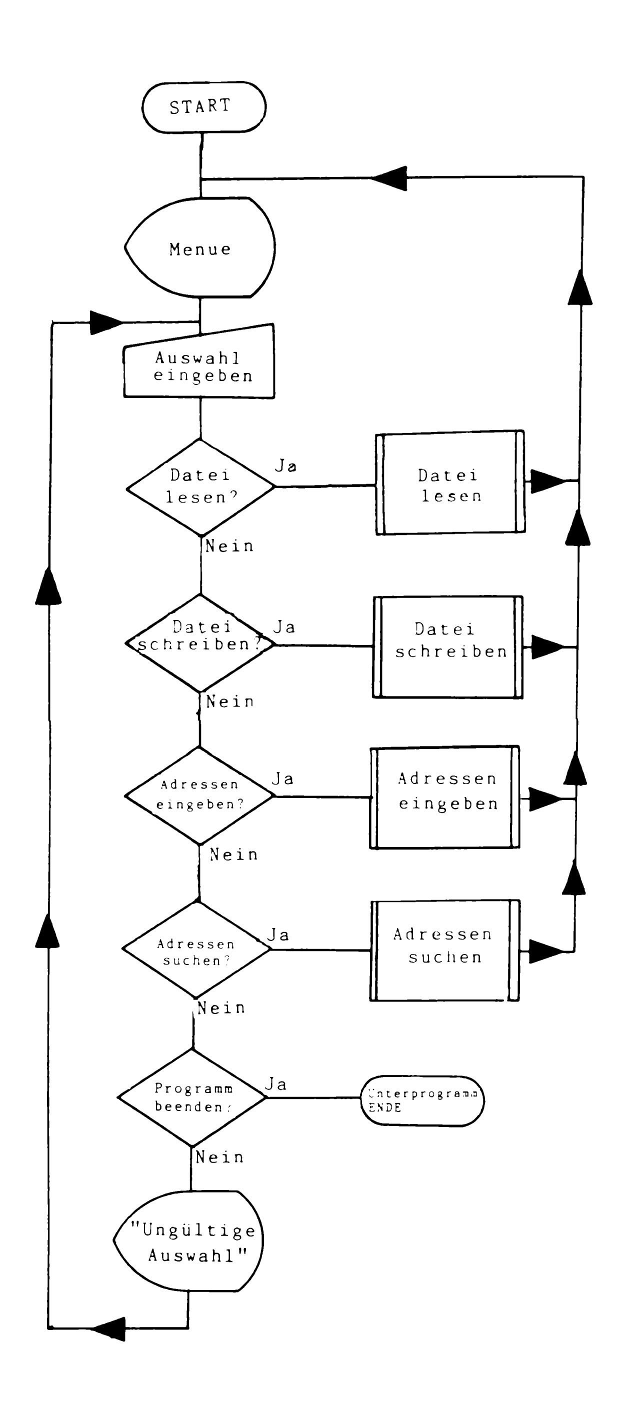
An unserer Adressenverwaltung soll die Anwendung eines Flußdiagrammes kurz demonstriert werden. Die Problemanalyse wurde im Kapitel 1.2. schon erstellt. Wir müssen uns jetzt nur noch Gedanken darüber machen, welche Teile der Gesamtproblematik als Unterprogramme ausgelegt werden sollen. Für unsere Adressenverwaltung könnte das folgendermaßen aussehen:

Hauptprogramm		Auswahlmenü und Eingabeüberprüfung.
Unterprogramm 1		Daten auf Diskette abspeichern.
Unterprogramm 2	:	Daten von Diskette einlesen.
Unterprogramm 3		Adressen eingeben
Unterprogramm 4	:	Adressen suchen.
Unterprogramm 4.1.	:	Liste nach Schlüssel ausgeben.
Unterprogramm 4.2.		Adressen nach Kriterium suchen.

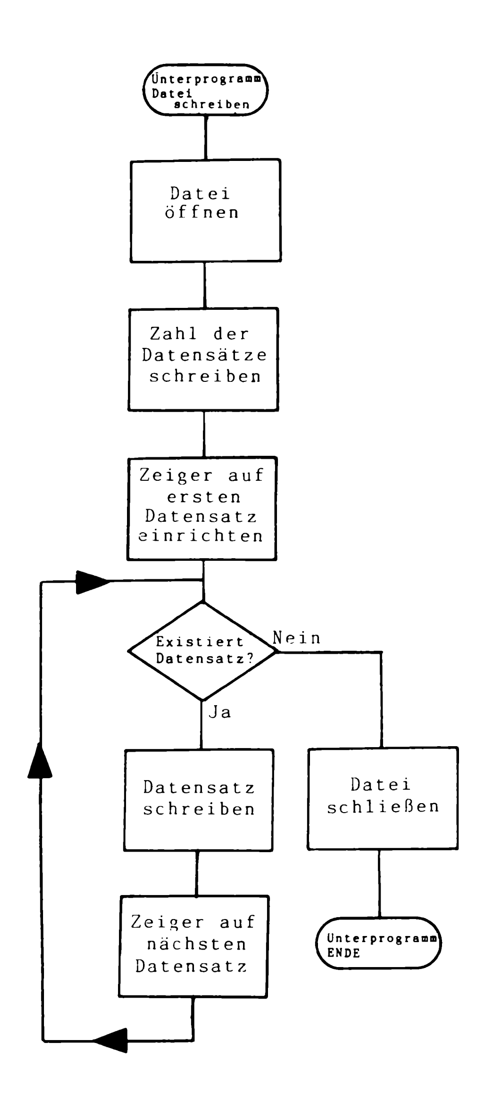
Es ist wenig sinnvoll, manchmal sogar unmöglich ein Gesamtflußdiagramm über die komplette Problematik mit allen programmtechnischen Raffinessen zu erstellen. Zuerst sollte ein sehr grober Ablaufplan der Problematil erstellt werden, der das Zusammenwirken zwischen Hauptprogramm und den einzelnen Unterprogrammen wiederspiegelt. Erst wenn die Flußdiagramme

für die - oben festgelegten - Teilprobleme erstellt werden, sollte man den Datenfluß näher beschreiben (untergeordnete Flußdiagramme).
Nachdem der gesamte Ablauf im Groben bekannt ist, werden nun die einzelnen Teilprobleme graphisch dargestellt. Wie oben schon erwähnt, wird hier auf den Datenfluß etwas näher eingegangen.

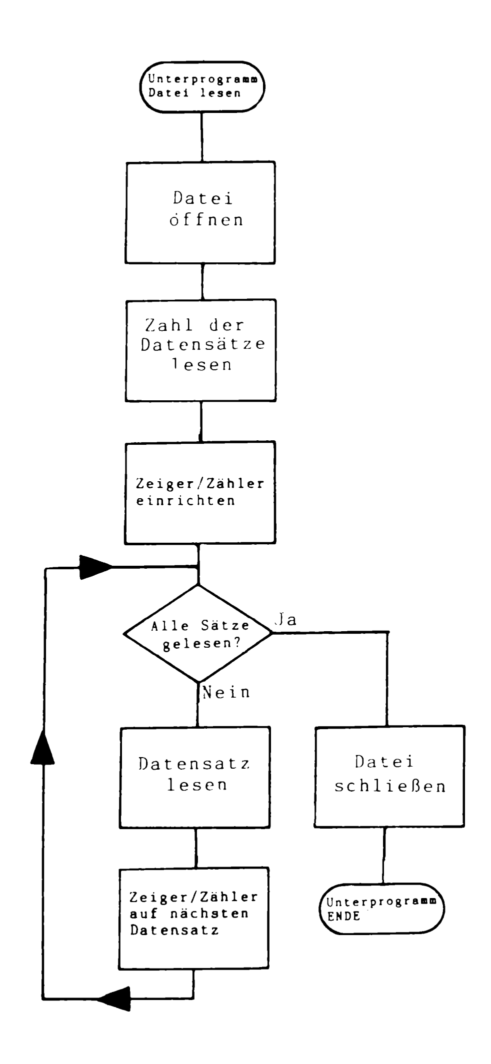
Hauptprogramm : Auswahlmenü und Eingabeüberprüfung.


START
Menue
Auswahl eingeben
Datei lesen?
Ja
Nein
Datei lesen
Datei schreiben?
Ja
Nein
Datei schreiben
Adressen eingeben?
Ja
Nein
Adressen eingeben
Adressen suchen?
Ja
Nein
Adressen suchen
Programm beenden?
Ja
Nein
Unterprogramm ENDE
"Ungültige Auswahl"

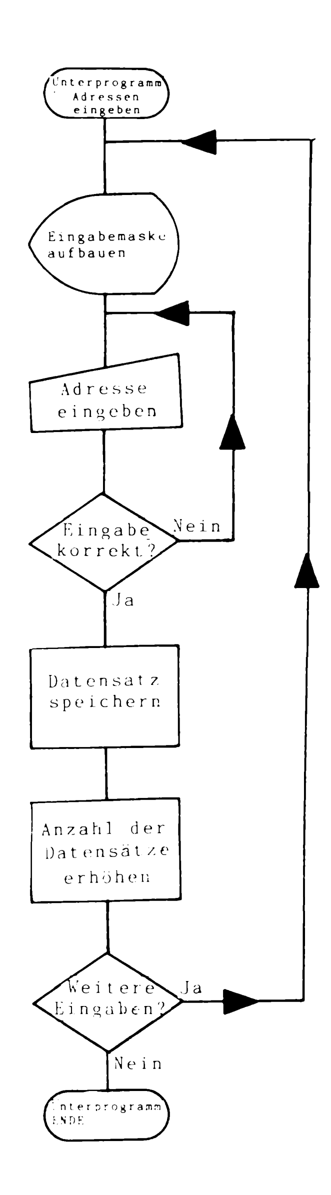
Unterprogramm 1: Daten auf Diskette abspeichern.



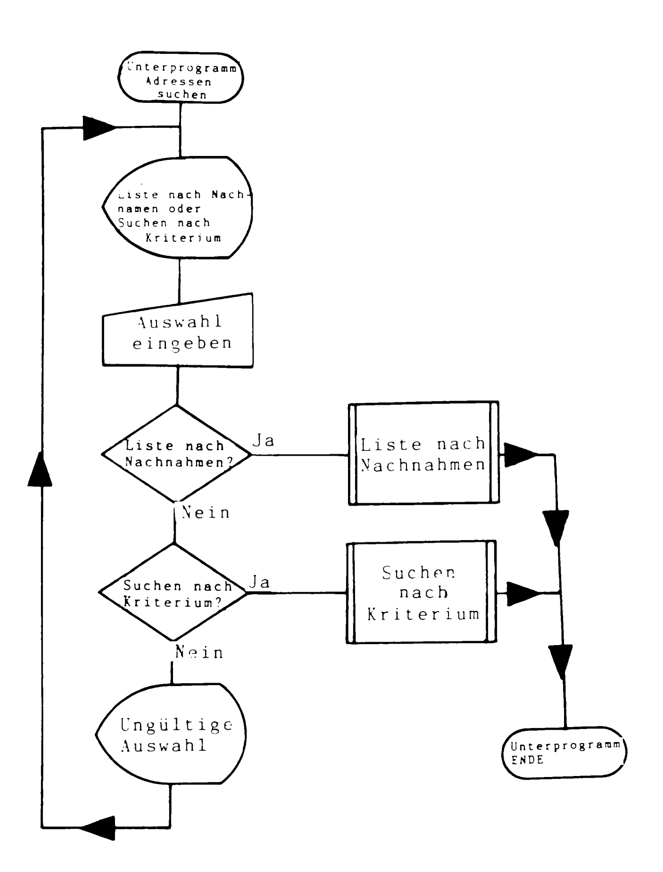
Unterprogramm 2: Daten von Diskette einlesen.



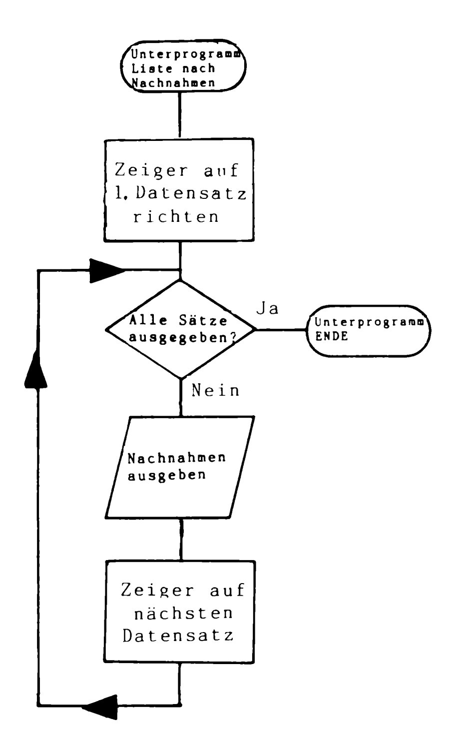
Unterprogramm 3: Adressen eingeben.



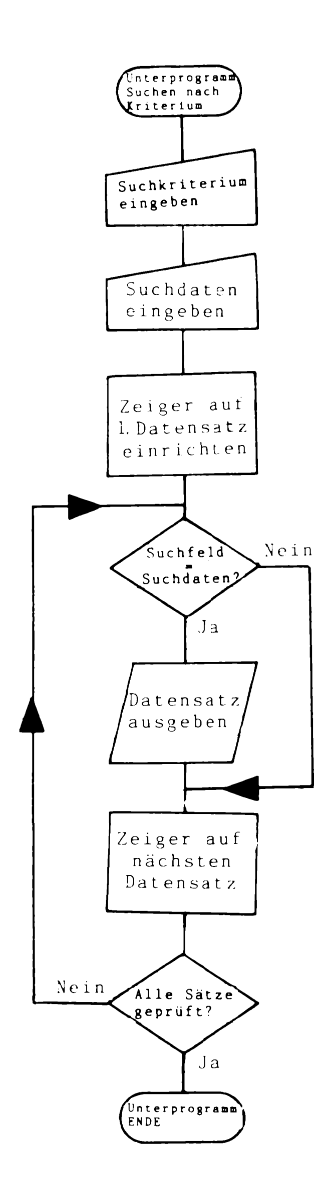
Unterprogramm 4: Adressen suchen.



Unterprogramm 4.1.: Liste nach Schlüssel ausgeben.



Unterprogramm 4.2.: Adressen nach Kriterium suchen.



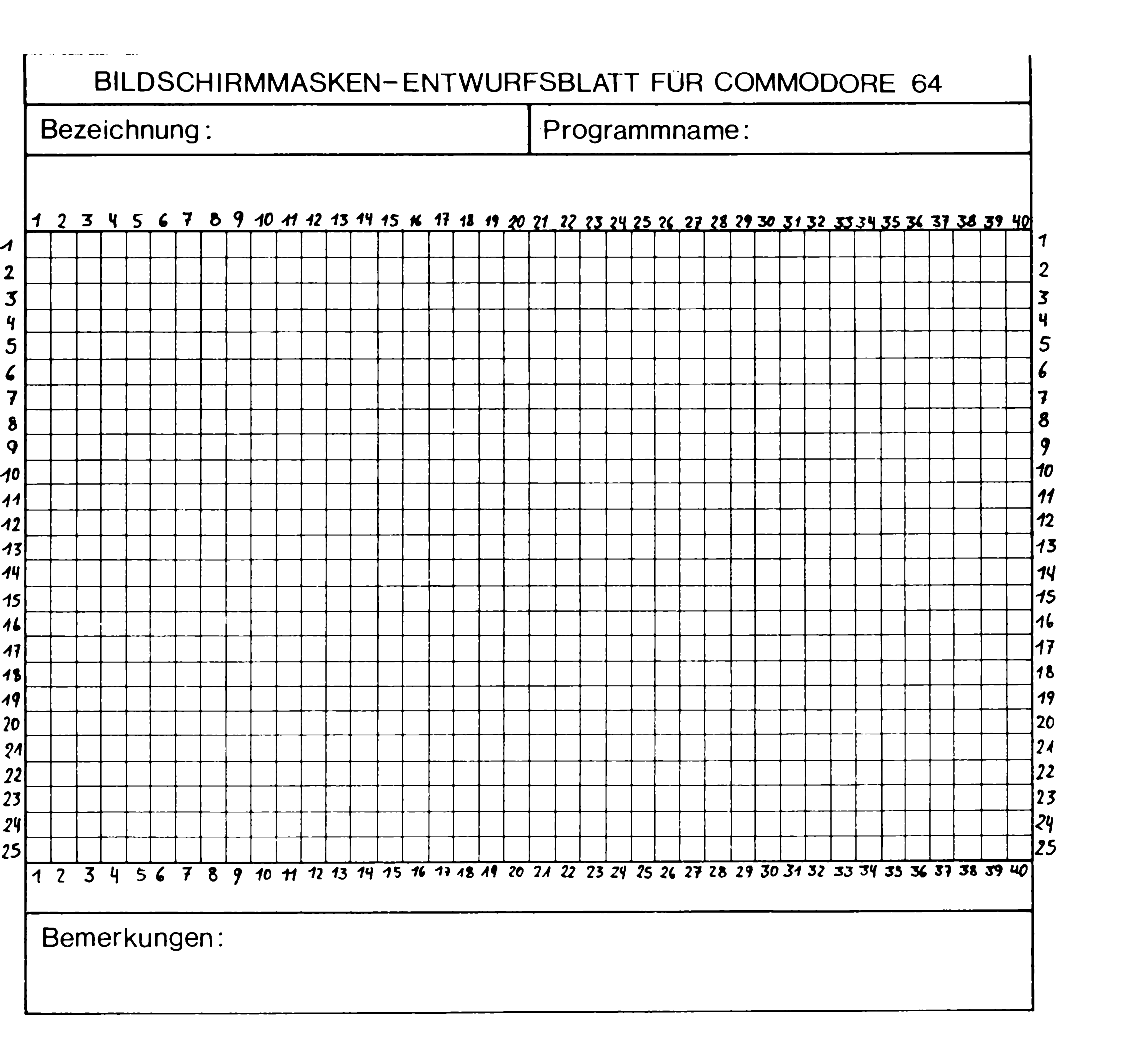
Nachdem nun all dies geschehen ist, kann man den Programmentwurf als abgeschlossen betrachten. Mit der eigentlichen Programmierung (Codierung) kann man jedoch immer noch nicht anfangen. Zuvor muß man weitere, detaillierte Überlegungen anstellen, die vom Entwurf der verschiedenen Bildschirmmasken bis hin zum eigentlichen Programmaufbau reichen sollten. Wie man hier am zwechmäßigsten vorgeht wollen wir Ihnen auf den folgenden Seiten zeigen.

1.4. Vorbereitungsarbeiten zur Programmierung

Nach der Definition der Programmidee bzw. der Istaufnahme der geplanten Anwendung und der Festlegung des späteren Programmablaufes können Sie noch längst nicht anfangen, zu programmieren. Zunächst sind noch einige sehr wesentliche Vorarbeiten zu erledigen. Alle diese Vorarbeiten, von der Gestaltung der Bildschirmmasken bis zu den grundsätzlichen überlegungen zum Programmaufbau sind sehr wichtig und sollten zeitlich unbedingt vor dem Beginn der eigentlichen Programmierung liegen. Auf diese Weise vermeiden Sie, auch wenn Ihnen viele dieser Vorarbeiten zunächst lästig und unnötig erscheinen mögen, spätere Mehrarbeit und verhindern mögliche Fehler.

1.4.1. Die Gestaltung der Bildschirmmaske

Mit der Bildschirmmaske und ihrem optischen Aufbau steht und fällt ein Programm. Je klarer und logischer Ihre Bildschirmmasken aufgebaut sind, desto einfacher wird die Bedienung des Programms für den späteren Benutzer. Schließlich ist die jeweilige Maske in der Regel das einzige, was er vom Programm sieht. über die sinnvolle Gestaltung einer Bildschirmmaske und die programmtechnische Abwicklung finden Sie detaillierte Aussagen im Kapitel 2.1..
Zunächst befassen wir uns hier mit der optischen Gestaltung der Bildschirmmasken. Als Hilfsmittel hierfür haben sich Bildschirmmasken-Entwurfsblätter sehr bewährt. In der Groß-EDV gehören sie zum Standard-Handwerkszeug eines Programmierers. Wir haben ein solches Bildschirmmasken-Entwurfsblatt für den COMMODORE 64 entwickelt und präsentieren es Ihnen nachfolgend. Im Grunde genommen ist es ein einfaches Abbild des 64er Bildschirms mit seinen 25 x 40 = 1000 Feldern. Kopieren Sie sich dieses Blatt beliebig oft und experimentieren Sie anschließend solange, bis Ihnen der Aufbau Ihrer Bildschirmmaske gefällt. Schnell werden Sie feststellen, daß sich mit Radiergummi und Bleistift leichter und schneller gestalten läßt als durch das fortlaufende Ändern von Programmzeilen.



BILDSCHIRMMASKEN-ENTWURFSBLATT FÜR COMMODORE 64
Bezeichnung:
Programmname:
Bemerkungen:

Mit einem fertigen Bildschirmmasken-Entwurfsblatt programmiert es sich hinterher erheblich leichter. Die Zeilen- und Spaltennummerierung des Entwurfsblattes hilft Ihnen bei der exakten Positionierung des Cursors in Ihrem Programm. Durch den endgültigen Aufbau Ihrer Bildschirmmaske erhalten Sie zudem wichtige Hinweise darüber, welche und wieviele Eingabe- bzw. Ausgabevariablen für die jeweilige Maske vorgesehen werden müssen. Dies kommt Ihnen besonders bei der Erstellung der Variablenliste (siehe Kapitel 1.4.4.) zugute.

1.4.2. Die Festlegung der Dateien

Wenn Sie in Ihrer Problemanalyse festgestellt haben, daß in Ihrem Programm Dateien benutzt werden, so sollten Sie jetzt den Aufbau dieser Datei(en) genau festlegen. Auch hierzu haben wir für Sie wieder ein entsprechendes Entwurfsblatt entwickelt, mit dem Sie wie bei den Bildschirmmasken Aufbau und Struktur jeder einzelnen Datei genau festlegen und beschreiben können. Wir haben das DATEIENTWURFSBLATT nachfolgend einmal als Muster ausgefüllt und einmal blanko zum beliebigen Kopieren abgebildet.

Hier die Erklärung der einzelnen Positionen des Dateientwurfsblattes:

Dateiname : Tragen Sie hier den Namen Ihrer Datei ein. Bei der Namenswahl sollten Sie einen Bezug zur Aufgabe der Datei wählen, z.B. ADRESS für eine Adressdatei oder FIBDAT für eine Parameterdatei zur Finanzbuchhaltung

Programmname : In dieses Feld gehört der Name des Programms, zu dem die Datei gehört

Dateiart : Geben Sie hier an, ob es sich um eine sequentielle, relativ, direkt oder ISAM bzw. QUISAM organisierte Datei handelt

Datensatzlänge : die Gesamtlänge eines Datensatzes

max. Anzahl Datensätze : geben Sie hier, falls die Dateiart eine feste Dateilänge vorschreibt oder Sie aus organisatorischen Gründen (die 170 K einer VC-1541 reichen nicht weit und steter Diskettenwechsel ist sehr lästig) den Umfang Ihrer Datei limitieren wollen

Nr.	: gibt die fortlaufende Nummer der Felder an und hat nur organisatorische Bedeutung
Art	: hier können Sie angeben, ob es sich um ein alphanumerisches (A), numerisches (N) oder gepacktes (P) Datenfeld handelt
Länge	: die maximale Länge des einzelnen Feldes
Position	: der Beginn des Feldes vom Satzanfang aus gesehen
Variable	: hier können Sie bereits die entsprechende Ein- bzw. Ausgabevariable festlegen. Dies trägt wesentlich zur späteren Übersichtlichkeit und Änderungsfreundlichkeit Ihres Programms bei und erleichtert die Erstellung der Variablenliste
Bemerkung	: hier können Sie z.B. gewisse Codes festlegen, wie z.B. Anredeschlüssel (0=Herr, 1=Frau, 2=Firma). In das zweite Feld Bemerkung am Fuß des Blattes gehören dann Dinge, die die gesamte Datei betreffen, wie z.B. die Bearbeitungsformen mit dem oben angegebenen Programm (ERFASSEN, AUSWERTEN etc.)
Zugriff von anderen Programmen	: häufig kommt es vor, daß Dateien von mehreren Programmen benutzt werden. Geben Sie hier an, welche Programme noch auf diese Datei zugreifen und zu welchem Zweck und in welcher Form diese Zugriffe geschehen. WICHTIG: Wenn eine Datei in mehreren Programmen verwendet wird, empfiehlt es sich, ihr in allen Programmen jeweils die gleichen Ein-/Ausgabevariablen zuzuordnen. Dies erhöht ungemein die Übersichtlichkeit und Änderungsfreundlichkeit eines Programmpaketes

DATEIENTWURFSBLATT FÜR COMMODORE 64		
Dateiname:	Programmname:	max. Anzahl Datensätze :
Dateiart :	Datensatzlänge:	

DATENSATZAUFBAU-einzelne Felder

Nr.	Bemerkung	Art	Länge	Position	Variable	Bezeichnung

Bemerkungen :	Zugriffe von anderen Programmen	
	Programmname	Zugriffszweck

DATEIENTWURFSBLATT FÜR COMMODORE 64

Dateiname: ADRESS Programmname: MEMBDATA max. Anzahl Datensätze : 300

Dateiart : REL Datensatzlänge: 76

DATENSATZAUFBAU-einzelne Felder

Nr.	Bemerkung	Art	Länge	Position	Variable	Bezeichnung
1	laufende Nr.	N	3	1	A1	Schlüssel
2	Vorname	A	12	4	A2$	
3	Name	A	15	16	A3$	
4	Status	N	1	31	A4	1= Vollzahler, 2= Jugendliche
5	Postleitzahl	N	4	32	A5	3: Student, 4= passives Mitgli
6	Ort	A	20	36	A6$	
7	Straße	A	20	56	A7$	
8	Beitrag	N	1	76	A8	0= Beitrag steht noch aus 1 = hat bezahlt

Bemerkungen:

Anlegen, Ändern, Löschen, Druck Mitgliederliste und Adressaufkleber

Zugriffe von anderen Programmen

Programmname	Zugriffszweck
MAHN	Druck Beitragsmahnungen

1.4.3. Rechenformeln

Lassen Sie uns jetzt zu einem Thema kommen, das nur zu häufig im Rahmen der Programmvorbereitung übergangen wird und dann für endlose Fehler sorgt. Gemeint sind alle in Ihrem Programm vorkommenden Rechenformeln, von der Mehrwertsteuer bis zur Rundungsroutine. Es empfiehlt sich, hier, wie nachfolgend beschrieben vorzugehen. Als Beispiel haben wir dabei die Errechnung einer Bruttoertragsprovision gewählt.

Vorgehen	Beispiel
1. Legen Sie zunächst fest, was überhaupt woraus errechnet werden soll	Die Bruttoertragsprovision (BP) soll sich errechnen als 10% der um einen Sockelbetrag von DM 500.- verminderten Differenz zwischen Einkaufspreis (EK) und Verkaufspreis (VK), wobei der Verkaufspreis vorher um die darin enthaltene Mehrwertsteuer zu verringern ist
2. Schreiben Sie als nächstes die entsprechende Formel in der ursprünglichen mathematischen Form auf	BP=(VK:1.14-500-EK)*0.1
3. Rechnen Sie zunächst diese Formel per Hand bzw. mit Hilfe eines Taschenrechners anhand einiger typischer Musterwerte durch, und prüfen Sie die Korrektheit Ihrer Formel	(5700:1.14-500-2500)*0.1=200

4. Setzen Sie jetzt Ihre Formel, falls sich sich als richtig erweist, in ein BASIC-Programm um. Dieses BASIC-Programm sollte nach Möglichkeit genauso aussehen, wie später in Ihrem Anwendungsprogramm. Dies gilt insbesondere für die zu benutzenden Variablen. Dies ist insbesondere wichtig, da bei einer Umsetzung sonst wieder Fehler passieren können. Beachten Sie bitte, woher die Grundwerte kommen und wohin das Ergebnis soll. Überprüfen Sie auch, ob diese Rechenformel noch häufiger in Ihrem Programm benötigt wird. Dies ist sowohl für die Variablenfestlegung wichtig, als auch für die eventuelle Gestaltung Ihrer Formel als ein mit GOSUB anzuspringendes Unterprogramm.

   BP = (V1:M1-500-EK)*0.1

   wobei:

   BP = Bruttoertragsprovision

   V1 = Verkaufspreis inkl. Mehrwertsteuer

   M1 = Mehrwertsteuerverteiler

   EK = Einkaufspreis

5. Testen Sie jetzt Ihr Programm mit den gleichen Werten wie in Schritt 3 und überprüfen Sie, ob die Ergebnisse identisch sind. Andernfalls korrigieren Sie Ihr Programm solange, bis es stimmt

Das Verfahren mag sich sehr umständlich anhören, erspart Ihnen aber im Endeffekt eine Menge Arbeit. Oft beruhen spätere Programmfehler auf fehlerhafte Rechenroutinen, wobei es sich oft um simple Rechenvorgänge handelt, die der Programmierer nach dem Motto "in den Grundrechenarten bin ich seit dem 2. Schuhljahr Spitze" salopp ohne Überprüfung codiert hat.

1.4.4. Die Druckausgabe

Zu den Programmvorbereitungen gehört in jedem Fall auch die Planung der Druckausgabe. Wie schon bei der Bildschirmmaske, sind hier auch optische und ergonomische Gesichtspunkte wichtig, da die Druckausgabe, die meist als Liste erfolgt, ebenfalls eine wichtige Verbindung zwischen dem Benutzer einerseits und dem Programm bzw. dem Computer andererseits darstellt. In der Regel gehen einer Druckausgabe mit Ausnahme eines einfachen Listings immer irgendwelche Aufbereitungen der auszudruckenden Daten voraus. Deshalb sollte die Druckausgabe sehr sorgfältig geplant werden. Bei der Festlegung, was in wlecher Form ausgedruckt werden soll, sind die folgenden Punkte in der angegebenen Reihenfolge zu berücksichtigen:

- was soll gedruckt werden

- welche Dateien werden benutzt und welche Datensätze oder Teile hiervon

- in welcher Form sollen die Daten vorher aufbereitet und ausgewertet werden. Bei Rechenformeln gehen Sie bitte analog wie in Kapitel 1.4.3. beschrieben vor

- auf welchen Drucker soll ausgedruckt werden.
  Der jeweilige Drucker mit seinen Möglichkeiten und Eigenschaften (Druckbild, Formularbreite, Zeichen pro Zeile, Anzahl der Durchschläge etc.) bestimmt wesentlich die Druckausgabe. In der Regel werden Sie nur einen Drucker haben. Schreiben Sie aber Programme für fremde Benutzer, so müsen Sie Ihre Druckausgabe so flexibel gestalten, daß sie auch auf unterschiedlichen Druckern ablaufen kann. Besonders gilt dies für Standardprogramme.

- wie soll die eigentliche Druckmaske aussehen
  Festgelegt werden muß hier nicht nur, welche Elemente

der einzelnen Datensätze in welcher Reihenfolge ausgedruckt werden sollen, sondern auch wie die gesamte äußere Form der Druckmaske aussehen soll. Hierzu gehören Breite, Durchschläge, einzeiliger/mehrzeiliger Druck, überschriften, Zwischenüberschriften, eventuelle Summen und anderes mehr. Gehen Sie am besten so wie bei der Bildschirmmaske vor, und legen Sie Ihre Druckmaske zunächst schriftlich mit Bleistift und Papier fest. Zwei wesentliche Unterschiede zur Bildschirmmaske müssen Sie dabei in jedem Fall berücksichtigen. Den Druckkopf können Sie im Gegensatz zum Bildschirmcursor nur in eine Richtung bewegen (Ausnahme: Plotter). Zeilen- und Spaltenposition des Druckkopfes können Sie während des Druckes nicht abfragen, sondern müssen stattdessen in Ihrem Programm mit entsprechenden Zählern arbeiten. Wichtig ist dies vor allem für den Vorschub bei Formularende.

- was passiert nach dem Ausdruck, z.B. Löschen einer Lagerbewegungsdatei nach Ausdruck der Lagerbewegungsliste.

Nützliche Routinen zur Gestaltung Ihrer Druckausgabe finden Sie in Kapitel 3.

1.4.5. Die Variablenliste und Tips zum Umgang mit Variablen

Da im COMMODORE-BASIC eine vorherige Definition von Variablen nicht erforderlich ist, ermuntert es zum sehr freigiebigen Umgang mit Variablen. Viele Programmierer streuen deshalb Variablen munter ohne Vorplanung quer durch ihr Programm. Die Folge sind nicht nur sehr unübersichtliche Programme, in denen sich - wenn überhaupt - schon nach kurzer Zeit auch der Ersteller selbst nicht mehr zurechtfindet. Es entstehen auch schwer zu lokalisierende Fehlerquellen, meist durch unbewußte Doppelbelegung von Variablen. Das Zusammenfügen von Programmen mit dem MERGE-Befehl oder gar die Arbeit mit der Overlaytechnik (Weitergabe von Variableninhalten an das nächste Programm, siehe nähere Darstellung im Kapitel 3) sind bei unkontrolliertem Variablengebrauch schier unmöglich.

Der Variableneinsatz muß also vernünftig geplant werden. Dies haben Sie ja schon in den vorhergehenden Kapiteln gesehen, in denen die Variablenfestlegung ein wichtiges Thema war. Hilfsmittel dazu ist eine Variablenliste. Sinnvollerweise sollte sie für jede benutzte Variable folgende Informationen enthalten:

```
E$ = Einkaufspreis
     erstmals benutzt in Zeile 650
     weiter in Zeilen 670, 830, 835, 1160
```

Die ersten beiden Zeilen des oben angeführten Beispiels, nämlich die Erklärung des Variableninhalts und die Angabe der Zeile, in der die Variable zum ersten Mal benutzt wird, sollten von vornherein in jeder Variablenliste enthalten sein. Die dritte Zeile mit der Nennung aller Zeilen, in denen die Variable anschließend noch vorkommt, ist zwar wünschenswert, aber mit größerem Aufwand verbunden. Deshalb kann diese Information auch nach Programmfertigstellung unter Zuhilfenahme eines Toolkits oder eines Cross Reference Listings zusammengestellt werden. In jedem Falle aber gehört sie zu einer guten Dokumentation und sollte später zur

abschließenden Testphase als wichtiges Hilfsmittel bei der Fehlersuche zur Verfügung stehen.

Nachfolgend dürfen wir Ihnen noch ein paar interessante Tips zum Umgang mit Variablen geben, die sich zwar teilweise gegenseitig ausschließen, aber für sich jeweils sehr nützlich sein können:

Gezielte Variablennamen schaffen Übersicht

Leider läßt das COMMODORE-BASIC keine langen Variablennamen zu. Aber auch zweistellige Variablennamen können Sie mit etwas Phantasie so wählen, daß sie Aufschluß über das geben, was sich hinter ihnen verbirgt. Bezeichnen Sie deshalb den Verkaufspreis ruhig mit VK, den Einkaufspreis mit EK, die Menge mit M und den Namen mit NA$.

Variablendefinition bringt Zeitgewinn

Der BASIC-Interpreter des COMMODORE 64 legt numerische Variable direkt hinter dem Programmende in Richtung RAM-Ende und Stringvariable jeweils vom RAM-Ende Richtung Programm nacheinander in der Reihenfolge der Erstnutzung ab. Bei jedem Variablengebrauch sucht er die abgelegten Variablen vom Anfang an durch, bis er die gewünschte gefunden hat. Rechtzeitige Definition häufig benutzter, zeitkritischer Variablen am Programmanfang (X$="", A=0 etc.) kann damit Zeitvorteile bringen.

Mehrfachverwendung spart Speicherplatz

Jede zusätzliche Variable belegt Speicherplatz. Wenn aber Speicherplatz knapp ist, bietet es sich an, bestimmte Variable mehrfach für unterschiedliche Aufgaben zu verwenden. Zum Beispiel Zählvariable bei Programmschleifen. Allerdings steht dieser Punkt der oft geforderten Übersichlichkeit Ihres Programms unter Umständen entgegen. Wenn mangelnder Speicherplatz Sie zu einer Mehrfachverwendung von Variablen zwingt, ist deshalb eine exakte Festlegung, welche Variable wann wofür verwendet wird, unbedingt notwendig.

Gleiche Variable für gleiche Anwendungen

Verwenden Sie möglichst in all Ihren Programmen für gleiche Anwendungen stets gleiche Variable. So schaffen Sie nicht nur Übersicht für sich selbst in Ihren Programmen, sondern ermöglichen sich auch den Aufbau und die Nutzung einer umfangreichen Bibliothek an Programmbausteinen, die sich gegenseitig nicht durch gleiche Variablennamen ins Gehege kommen.

DIMensionierung spart Speicherplatz

Indizierte Variable vom Typ A$(1) braucht man im COMMODORE-BASIC erst bei mehr als 10 Einträgen vorher zu dimensionieren. Allerdings reserviert dafür der Interpreter beim Gebrauch einer nicht DIMensionierten Variable 10 Speicherplätze. Durch entsprechende DIMensionierung, z.B. A$(4), können Sie dies umgehen und den benötigten Speicherplatz auf das unbedingt gebrauchte beschränken.

1.4.6. Die Programmdokumentation

Bevor wir über die Dokumentation sprechen, sollten wir erst einmal zwei Dinge voneinander trennen, die immer wieder durcheinandergeworfen werden. Die Bedienungsanleitung, zu der wir in einem späteren Kapitel noch kommen, hat nichts mit der Programmdokumentation zu tun. Sie ist für den Benutzer eines Programms geschrieben und erklärt ihm den Umgang mit diesem Programm. Die Dokumentation hingegen schreiben Sie in erster Linie für sich selbst, den Ersteller des Programms, und für alle anderen, die Ihr Programm einmal pflegen oder ändern sollen. Die Dokumentation besteht aus folgenden Teilen:

- Programmidee und Programmablaufplan

- Bildschirmmasken-Entwurfsblätter

- Dateientwurfsblätter

- Rechenformeln

- Druckmasken

- Variablenliste

- zeilenorientierte Programmbeschreibung

Sie sehen, bei vernüftiger Programmplanung steht der größte Teil der Dokumentation bereits vor dem ersten Programmierbefehl. Was jetzt nur noch fehlt, ist die zeilenorientierte Programmbeschreibung. In ihr sollte lückenlos aufgeführt sein, was wo passiert. Voraussetzung dafür ist allerdings, daß Sie diesen wichtigen Teil der Dokumentation paralell zur eigentlichen Programmierung erstellen. Niemand kennt Ihr Programm so gut wie Sie selbst im Moment der Programmierung. Ersparen Sie sich deshalb spätere Frustrationen durch inzwischen unverständlich gewordene Listings, und dokumentieren Sie Ihr Programm von Anfang an. REM-Zeilen sind mit Sicherheit eine gute Ergänzung, aber keine Alternative zur zeilenorientierten Programmbeschreibung.

Wenn Sie viel mit Programmbausteinen arbeiten, empfiehlt es sich, die detaillierten Beschreibungen dieser Bausteine per Textverarbeitung zu verwalten. Dann können Sie Ihre Dokumentation genauso zusammenMERGEn, wie Ihr Programm.

1.4.7. Grundsätzliche Überlegungen zum Programmaufbau

Dies ist der letzte Schritt vor der eigentlichen Programmierung (Codierung). Hier sollten Sie grundlegende Überlegungen bezüglich des Programmaufbaus anstellen. Dabei geht es diesmal nicht so sehr um den späteren Benutzer des Programms, sondern um Sie selbst, den Ersteller. Ein einwandfreier, übersichtlicher Aufbau Ihres Programms erleichtert nicht nur die Programmierung, sondern auch die Testphase und vor allem spätere Änderungen und Erweiterungen.

Sicherlich haben Sie sich, bevor Sie mit den umfangreichen Programmvorbereitungen begannen, bereits Gedanken über Programmgröße und Speicherkapazität sowohl im COMMODORE 64 als auch auf der Floppy gemacht. Es wäre wohl völlig unsinnig, Projekte wie etwa eine Mitgliedsdatei für ADAC oder AOK auf einem COMMODORE 64 überhaupt anzugehen. Aufgrund der vielen zusätzlichen Informationen und Daten, die Sie im Verlauf der Programmvorbereitung gesammelt haben, können und sollten Sie jetzt aber abschätzen, inwieweit Sie von vornherein großzügig mit Speicherplatz umgehen können, was natürlich die Programmerstellung leichter macht und sowohl Übersichlichkeit als auch Änderungsfreundlichkeit fördert, oder ob Sie grundsätzlich mit Speicherplatz sehr sparsam umgehen müssen. Diese Überlegungen haben wesentlichen Einfluß auf die spätere Programmierung und den Programmaufbau. Bitte übergehen Sie diesen Punkt nicht. Ein zu großes Programm später so zurechtzuzimmern, daß es gerade in den Speicher paßt, ist eine wenig dankbare und sehr aufwendige Aufgabe, die Sie sich ersparen sollten

Als nächstes sollten Sie untersuchen, inwiefern Sie auf fertige Bausteine oder Teile anderer Programme zurückgreifen können. Durch Übernahme derartiger Programmteile können Sie sich viel Arbeit ersparen. Grundsätzlich sollten Sie deshalb die Zeilennummern und Variablenbezeichnungen häufig verwendeter Unterprogramme so wählen, daß diese Unterprogramme weder miteinander noch zu möglichen neuen

Programmen in Konflikt stehen. Neue Programme lassen sich damit ohne Umsetzprobleme rasch aus vorhandenen Modulen zusammensetzen.

Legen Sie nun die Gliederung von Hauptprogramm und Unterprogrammen endgültig fest und vergeben Sie im vorhinein die Zeilennummern bzw. Zeilenbereiche, wo das betreffende Programmteil im Listing zu plazieren ist.

Wo sollen im Programm welche REM-Zeilen vorgesehen werden ? Dieser Frage sollten Sie hier auch nachgehen. Trennen Sie die einzelnen Programmteile durch REM-Zeilen und geben durch entsprechende Kommentare Hinweise darüber, welche Aufgabe das jeweilige Programmodul zu erfüllen hat. Eine wichtige REM-Zeile sollte ganz zu Angfang Ihres Programms stehen:

```
10 REM KONTOVERWALTUNG, VERSION 2.06, STAND 10.12.1984
```

diese Zeile sollten Sie stets aktualisieren, damit Sie immer wissen, mit welcher Version Ihres Programms Sie es zu tun haben. Im Anschluß daran sollte dann in einer REM-Zeile der ebenfalls sehr wichtige Copyright - Vermerk folgen.

Schätzen Sie ab, wie groß Ihr zukünftiges Programm voraussichtlich werden wird. Sicherlich sind solche Schätzungen nicht sehr zuverlässig, man kann jedoch recht gut abwägen ob Programm und vorgesehene Datenmenge im Arbeitsspeicher Platz finden.

Die Programmvorbereitung hat nun Ihren Abschluß gefunden. Bleibt noch, das Programmlisting zu erstellen und in den COMMODORE 64 einzugeben.
Vergessen Sie dabei nicht, in sehr regelmäßigen kurzen Abständen Ihr Programm oder was bisher davon steht abzuspeichern. Sie wären nicht der Erste, der besessen von stets neuen Geistesblitzen wie wild drauflosprogrammiert, daß Abspeichern vergißt und dann irgendwann mitten in der Nacht übermüdet den Rechner ausschaltet (oder sich am nächsten

Morgen von seinem kleinen Sohn das Kabel herausziehen läßt). Auch hochkarätigen Experten ist soetwas schon passiert. Speichern Sie deshalb in kurzen Abständen Ihr Programm ab und fertigen Sie regelmäßig Sicherheitskopien an.

1.5. Strukturiertes Programmieren an einem Beispiel - Der COMMODORE 64 als Taschenrechner

Strukturiertes Programmieren. Dieses Schlagwort hört man in letzter Zeit immer häufiger in den branchenorientierten Massenmedien. Was ist eigentlich strukturiertes Programmieren ?

Wenn Sie in der Programmvorbereitung Ihr Problem formuliert und in kleinere Teilprobleme zerlegt haben, ist der erste Schritt zu einer strukturierten Programmierung bereits getan. Hier haben Sie bereits gedanklich festgelegt, welche Struktur Ihr späteres Programm einmal haben soll. In der Regel wird man später diese Teilprobleme (soweit sinnvoll) als Unterprogramme deklarieren. Selbst dann, wenn Sie keine Unterprogramme verwenden wollen ist die Aufteilung der Teilprobleme in einzelne Blocks (Programmmodule) erforderlich. Ein strukturiertes Programm ist immer sehr übersichtlich und leicht nachvollziehbar. Leider wird nicht überall von dieser wirkungsvollen Technik Gebrauch gemacht.

Oftmals hört man von Verfechtern anderer Programmiersprachen als BASIC (z.B. PASCAL-Programmierern), daß eine strukturierte Programmierung in BASIC nicht möglich sei. Sicherlich kann man in PASCAL ein Programm wesentlich besser durchstrukturieren als in BASIC, was jedoch nicht bedeutet, daß ein strukturiertes Programmieren in BASIC unmöglich ist. Die BASIC-Sprache bietet uns Statments, mit denen eine einfache Strukturierung durchaus möglich ist. Doch hierzu kommen wir später noch. Zuerst wollen wir uns mit den Grundlagen dieser Technik eingehend befassen.

1.5.1 Vorbereitungen

Die Programmvorbereitungen sind fast die gleichen die wir schon in diesem Kapitel unter 1.2 - 1.4 beschrieben haben. Hier müssen Sie Ihr Problem jedoch viel feiner und detaillierter formulieren. Zerlegen Sie bitte Ihr Problem in kleine Teilprobleme. Es empfiehlt sich, jedes dieser Teilprobleme einzeln zu bearbeiten. Um Ihnen die Entwicklung eines strukturierten Programmes praktisch zu demonstrieren, haben wir für Sie noch ein kleines Beispielprogramm vorbereitet. An Hand dieses Beispieles wollen wir mit Ihnen Schritt für Schritt einen Programmentwurf anfertigen.

1.5.2 Die Programmanalyse

Auch hier wollen wir, wie unter 1.2 beschrieben, unsere Programmidee ausarbeiten und festlegen, was das Programm alles können soll. Für unser Beispielprogramm haben wir uns folgende Problemstellung einfallen lassen:

Der Commodore 64 soll als elektronischer Taschenrechner mißbraucht werden. Auf dem Bildschirm wird mit den, im Zeichensatz enthaltenen Graphikzeichen ein einfacher Taschenrechner dargestellt. Unser kleiner Rechenkünstler soll folgende Tastatur erhalten:

I. Die Zahlentasten 0-9

II. Tasten für die mathem. Operationen +,-,x und :

III. Die Taste '=' zur Ausführung einer Rechnung

IV. Die Taste 'C' zum Löschen der letzten Eingabe

V. Die Taste 'T' zum Löschen aller Register (ALL CLEAR)

VI. Die Taste 'E' um Zahlen als 10ner-Potenz darstellen zu können

VII. Die Taste für den Dezimalpunkt (.)

VIII. Die Taste 'S' zum Ausschalten des Rechners

Die jeweilige Taste wird durch Drücken von >>RETURN<< betätigt. Damit man weiß, welche Taste im Moment aktuell ist, soll diese invers dargestellt werden. Dieser inverse Sektor muß mit den Cursor-Steuertasten am Computer beeinflußt werden können.

Alle Eingaben werden im Display unseres graphischen Taschenrechners angezeigt (max. Länge = 18 Stellen) und können im Bereich von -2.93873588E+38 bis 1.70141183E+38 liegen. Mit dem Rechner kann immer nur ein Zahlenpaar verarbeitet werden. Bei einem Eingabefehler wird das Programm mit einer entsprechenden Fehlermeldung unterbrochen. Beim Drücken einer Operationstaste (+,-,:,x) wird diese an der äußerst linken Stelle im Display angezeigt. Hierbei wird die vorhergehende Eingabe gelöscht. Nach Eingabe von '=' wird das Ergebnis errechnet und im Display ausgegeben.

Als Besonderheit wird die zweite Eingabezahl als Konstante gespeichert. Durch widerholtes Betätigen von '=' wird der angezeigte Ergebniswert mit dieser Konstanten weiterverarbeitet. Um die Bedienung unseres Rechners zu erleichtern, werden am rechten Bildschirmrand die wichtigsten Tasten erläutert (C,T,S,E).

Das Programm wird in folgende Teilabschnitte gegliedert:

A) Rücksetzen aller Variablen
B) Erzeugen der Bildschirmgraphik
C) Ausgangsposition invers darstellen (U)
D) Tatstaturabfrage mit GET
E) Auswertung der eingegebenen Zeichen (U)
F) Displayanzeige (U)
G) Steuerung durch CRSR-Tasten (U)
H) Löschen bei Betätigen von 'C' (U)
I) Löschen bei Betätigen von 'T' (U)

J) Operationstaste abspeichern und ausgeben (U)

K) Rechenoperationen ausführen (U)

L) Display vollständig löschen (U)

M) Display löschen (erste Stelle bleibt stehen) (U)

Alle mit (U) gekennzeichneten Teilprobleme werden als Unterprogramm deklariert, so daß das Hauptprogramm nur aus den Punkten A,B und D besteht.

1.5.3 Die Variablenliste

Folgende Variablennamen sollen im Programm verwendet werden:

X	=>	Wert der aktuellen Bildschirmspeicheradresse
DY	=>	Zeichencode der Taste, die gerade betätigt wurde
I	=>	Zählvariable für Schleifen
FLAG	=>	Statusflag für Konstantenoperationen
RES	=>	Ergebnis der jeweiligen Rechenoperation
Q$	=>	Variable für GET - Abfrage
Z1$	=>	Erste Eingabezahl
Z2$	=>	Zweite Eingabezahl
SU$	=>	Speichervariable für Displayanzeige
OP$	=>	Aktuelles Rechenzeichen (+,-,x oder :)

1.5.4 Der Programmablauf

An dieser Stelle wartet sehr viel Arbeit auf Sie. Alle festgelegten Teilprobleme müssen hier zu einem logischen Ablaufmuster geordnet und einzeln bearbeitet werden. Bei der Programmplanung für unser Taschenrechnerprogramm haben wir im Abschnitt 1.5.2 die Teilprobleme bereits so geordnet, daß sie für diesen Schritt direkt übernommen werden können.

Stellen Sie Ihren Algorhytmus so auf, daß der BASIC-Befehl 'GOTO' n i c h t verwendet werden muß. Dieser Befehl ist einer strukturierten Programmierung nicht gerade förderlich. Wenn Sie innerhalb Ihres Programmes zu einem anderen Programmteil verzweigen wollen, verwenden Sie bitte nur bedingte Verzweigungsbefehle,wie zum Beispiel IF.....THEN (wie Sie zweigungsbefehl).

Unser Programm soll folgendermaßen aufgebaut sein:

1. Rücksetzen der Variablen mit CLR.
2. Erzeugen der Bildschirmgraphik: Es werden nur Graphikzeichen des Standardzeichensatzes verwendet.
3. Aufruf des Unterprogrammes zur Darstellung der inversen Ausgangsposition. Vor dem Aufruf durch GOSUB muß die Variable X auf ihren Anfangswert gesetzt werden (X=1868).
4. Über GETQ$ wird nun die Tastatur abgefragt. Das Programm bleibt hier so lange stehen, bis ein gültiges Zeichen eingegeben wird. Wurde eine entsprechende Taste betätigt, wird das Unterprogramm zur Auswertung der jeweiligen Eingabe aufgerufen.
5. Aufruf des Auswertemoduls mit GOSUB.
6. Hier läßt sich der GOTO-Befehl nicht umgehen (GOTO 4.), da das Programm zur erneuten Tastaturabfrage verzweigen

muß.

7. Alle weiteren Unterprogramme werden von dem Programmodul 'Eingabeüberprüfung' zentral eingeleitet.

Aufbau der Unterprogramme:

Im Folgenden wird der Aufbau der einzelnen Unterprogramme beschrieben. Wenn eines der beschriebenen Programmodule selbst Unterprogramme benutzt, wird dies im Text durch (U) kenntlich gemacht.

1. Unterprogramm: Auswertung der Eingabe

Der Variablen DY wird hier der Zeichencode des aktuellen Zeichens zugewiesen. Da dieser Wert über PEEK(X) direkt aus dem Bildschirmspeicher gelesen wird ist zu beachten, daß die Zeichencodes von alphanumerischen Zeichen n i c h t dem ASCII-Code entsprechen. Die gültigen Zeichencodes für den Bildschirmspeicher entnehmen Sie bitte Ihrem Handbuch (Anhang E, Seite 132ff).

Durch fünf IF-Abfragen wird nun getestet, welche Funktionstaste eingegeben wurde:

a) Betätigung von Taste 'C' (U)
b) Betätigen der Taste 'T' (U)
c) Rechenzeichen wurde eingegeben (U)
d) Gleichheitszeichen wurde eingegeben (U)
e) Betätigen der Taste 'S' (Programm wird hier beendet)

Wurde keines der oben getesteten Zeichen eingegeben, kann das eingegebene Zeichen (Ziffer) auf dem Display ausgegeben werden (U).

Rücksprung ins Hauptprogramm (RETURN) um eine neue Tastatur-

abfrage einzuleiten.

Dieses Programmodul wird immer dann aufgerufen, wenn die RETURN-Taste gedrückt wurde.

Für dieses Unterprogramm werden die Zeilennummern 1000-1500 vorgesehen.

2. Unterprogramm: CRSR UP wurde gedrückt

Wenn der Inhalt der Variablen X kleiner als 1429 ist, wird das Unterprogramm beendet (RETURN). Dieser Wert muß getestet werden, weil wir vermeiden wollen, daß der Cursor aus dem vorgesehenen Bereich 'ausbricht'.

Ist X nicht kleiner als 1429, wird die Variable X um 160 vermindert. Dadurch wird erreicht, daß der Cursor um 4 Zeilen nach oben springt. Durch einen POKE-Befehl wird das Zeichen, wo sich der Cursor vorher befand, vom inversen in den normalen Zustand gepoked.

Nun kann das neu angewählte Zeichen (Taste) invers dargestellt werden (U). Danach kann die Routine mit RETURN beendet werden.

Diese Routine wird durch das Unterprogramm : Welche Taste... aufgerufen.

Hier werden die Zeilen 2000-2500 reserviert.

3. Unterprogramm: CRSR DOWN wurde gedrückt

Die Funktion dieses Programmoduls ist im Prinzip die gleiche wie unter 2. beschrieben. Nur durch zwei Unterschiede heben sich beide Programmteile voneinander ab:

I. Es wird zuerst überprüft, ob der Inhalt der Variablen X größer als 1864 ist, um im 'JA'-Falle das Unterprogramm durch RETURN zu beenden.

II. Die Variable X wird nun um den Wert 160 erhöht (vier Zeilen nach unten).

Ansonsten weist dieses Unterprogramm den gleichen Algorhyt-mus auf wie das Programmodul 2.

Es werden die Zeilen 3000-3500 vorgesehen.

4. Unterprogramm: CRSR RIGHT wurde gedrückt

Nach dem Drücken von CRSR RIGHT muß zuerst überprüft werden, ob der Cursor nicht schon in seiner äußerst rechten Position steht. Hierfür wird eine Zählschleife benutzt (Variable I), die zeilenweise den jeweiligen Grenzwert in der Variablen I ablegt (STEP 40). Nun kann in einer IF - Abfrage, die in dieser Schleife eigebettet ist, getestet werden, ob X gleich I ist. Wenn 'JA', wird das Programm mit RETURN abgebrochen.

Da auf unserem graphischen Taschenrechner die Funktions-tasten von den Zifferntasten etwas abgesetzt sind, ist eine zweite Testschleife nötig die ermittelt (IF-Abfrage), ob der CURSOR 4 oder 5 Zeichen überspringen soll (unser Cursor soll ja nicht zwischen den Tasten erscheinen).

Werden beide Schleifen voll durchlaufen (kein Grenzwert), so wird die Variable X um 4 erhöht. Im Anschluß daran kann die alte Position des Cursors gelöscht (POKE) und die Neue dargestellt werden (U).

Wenn dies alles erfüllt ist, wird das Programmodul wieder verlassen (RETURN).

Der Aufruf dieser Routine erfolgt von dem Unterprogramm : 'Welche Taste...'.

Hier werden die Programmzeilen 4000-4500 reserviert.

5. Unterprogramm: CRSR LEFT wurde gedrückt

Dieses Programm verwendet exakt den selben Algorhytmus wie Programmodul 4. Da hier die Grenzwerte etwas anders liegen, nfdern sich lediglich die Werte der Variablen und Schleifen.

Dieses Programmteil soll in den Zeilen 5000-5500 stehen.

6. Unterprogramm: Taste 'C' wurde gedrückt

Hier wird die letzte Eingabe gelöscht. Zu diesem Zweck werden zuerst die Variablen SU$ und FLAG rückgesetzt (0 oder Leerstring). Anschließend wird das Display gelöscht (U).

Nun kann das Programm per RETURN wieder verlassen werden.

Aufgerufen wird dieses Programmodul durch die Routine 'Auswertung der Eingabe'.

Es werden die Zeilennummern 6000-6050 reserviert.

7. Unterprogramm: Taste 'T' wurde gedrückt

Alle Speichervariablen werden zurückgesetzt (SU$,OP$,Z1$, Z2$,RES,FLAG). Zum Rücksetzen der Stringvariablen SU$ wird das Unterprogramm 6 aufgerufen.

Programmende durch RETURN.

Hier werden die Zeilen 6100-6050 reserviert.

8. Unterprogramm: Operator wurde gedrückt

Das eingegebene Rechenzeichen wird in der Variablen OP$ abgespeichert (CHR$(DY)) und über einen PRINT-Befehl auf dem Display an der äußerst linken Stelle ausgegeben. Nun wird der Variablen Z1$ die bisher eingegebene Zahl als String zugewiesen. Danach wird das Display gelöscht (U) und die Routine beendet.

Der Aufruf dieser Routine erfolgt durch das Programmodul 'Auswertung der Eingabe'.

Es werden die Zeilen 6200-6250 vorgesehen.

9. Unterprogramm: Operation ausführen

Dieses Unterprogramm wird immer dann aufgerufen, wenn die

Gleichheitstaste ('=') gedrückt wurde.

Zunächst wird der Variablen Z2$ der Eingabewert zugewiesen. Nun wird das Gleichheitszeichen auf unserem Display (äußerst links) ausgegeben.

In einer Reihe von IF-Abfragen wird ermittelt welche Rechenoperation durchgeführt werden soll. Nun kann das Ergebnis mit der entsprechenden Formel augerechnet werden. Danach wird das Display gelöscht (U) und das Ergebnis wird ausgegeben (U).

Unterprogrammende durch RETURN.

Vorgesehene Zeilennummern: 6300-6500

10. Unterprogramm: Display löschen

Das gesamte Anzeigedisplay wird gelöscht. Hierzu wird eine Zählschleife verwandt, in der ein POKE-Befehl die entsprechenden Bildschirmadressen (1188-1207) mit dem Leerzeichen (32) belegt.

Der Unterprogrammaufruf erfolgt von den Programmodulen 6 und 7.

Vorgesehene Zeilennummern: 6900-7000

11. Unterprogramm: Zeichen invers darstellen

Wie Sie sicherlich wissen kann man jedes Zeichen invers darstellen. Man muß dem jeweiligen Zeichencode lediglich den

Wert 128 hinzuaddieren. Dies realisieren wir durch einen POKE - Befehl.

Der Aufruf erfolgt durch die Programmodule 2,3,4,5 und durch das Hauptprogramm.

Vorgesehene Zeilennummern: 10000-10100

12. Unterprogramm: Welche Taste wurde gedrückt

Dieses Unterprogramm wird durch das Hauptprogramm nach der Tastaturabfrage aufgerufen. Hier wird geprüft ob RETURN oder eine Cursorsteuertaste gedrückt wurde. Bei Eingabe einer Cursorsteuertaste wird das entsprechende Programmodul aufgerufen, während nach Eingabe von RETURN immer das Unterprogramm 1 selektiert wird.

Rückkehr zur Tastaturabfrage mittels RETURN.

Vorgesehene Zeilennummern: 20000-20050

13. Unterprogramm: Display-Ausgabe

Der Stringvariablen SU$ wird die anzuzeigende Zahl zugewiesen. Danach wird die Variable Z auf ihren Anfangswert gesetzt (1207). Nun wird die Länge von SU$ ermittelt und nötigenfalls korrigiert (max. 18 Zeichen).

Ein POKE-Befehl, der in eine Zählschleife eingebunden ist, poked die einzelnen Zeichen aus SU$ an die jeweiligen Bildschirmstellen. Ist die Schleife durchlaufen, wird Z wieder auf 1207 gesetzt bevor die Routine mit RETURN beendet wird.

Vorgesehene Zeilennummern: 20100 - 20300

14. Unterprogramm: Nur Operator bleibt stehen

Hier wird das Display nur Teilweise gelöscht, d. h. die linke Stelle für den Operator (+,-,x,:,=) bleibt stehen. Der Algorhytmus ist mit dem des Unterprogrammes 10 identisch.

Vorgesehene Zeilennummern: 20500 - 20600

Wir sind nun am Ende unserer Unterprogrammbeschreibung angekommen. Durch Angabe der geplanten Zeilennummern haben wir bereits die grobe Struktur unseres Programmes festgelegt. Sicherlich konnten wir durch die Beschreibung unseres Programmbeispieles wichtige Gedankenstützen für Ihre eigene Entwicklung geben und hoffen, daß Sie das hier gezeigte für eigene Probleme sinnvoll umsetzen können. Vielleicht erscheint Ihnen unsere Vorgehensweise etwas zu ausführlich, Sie werden jedoch sehen, daß sich diese Methode bei der weiteren Programmplanung auszahlt.

Nachdem der schriftliche Teil der Programmvorbereitung abgeschlossen ist können wir daran gehen, unser zukünftiges Programm graphisch darzustellen. Bei der strukturierten Programmierung ist es vielleicht sinnvoll, neben der üblichen Darstellung durch ein Flußdiagramm eine weitere Darstellungsform zu wählen: Das Struktogramm

Auf den nächsten Seiten wollen wir uns einmal intensiv mit diesem graphischen Hilfsmittel beschäftigen.

1.6. Das Struktogramm

Struktogramme oder auch Nassi-Shneidermann oder kurz NS-Diagramme genannt, sind sehr nützliche Hilfsmittel bei der Programmdarstellung in der strukturierten Programmierung. Gegenüber den Flußdiagrammen nach DIN 66001 hat man hier den Vorteil, auf Datenflußlinien und Richtungspfeile verzichten zu können. Der Datenfluß wird immer so dargestellt, daß man den Programmablauf von oben nach unten verfolgen kann. Durch die einheitliche, rechteckige Form der angewandten Symbole kann man den gesamten Datenflußplan ineinanderschachteln und so sehr viel Platz sparen. Im Folgenden wollen wir Ihnen die üblichen Symbole vorstellen. Gleichzeit werden wir Ihnen die entsprechenden Realisierungen mit Symbolen nach DIN66001 zeigen, damit Sie eine direkte Vergleichsmöglichkeit haben. Hier nun die Zeichen:

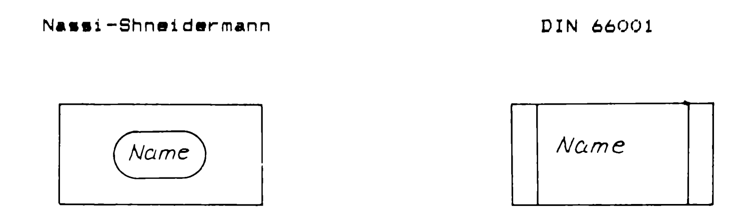
1. Der Verarbeitungsschritt

Ein einfaches Rechteck stellt einen Verarbeitungsschritt dar (genauso wie nach DIN 66001). Innerhalb dieses Rechteckes kann im Klartext eine entsprechende Anweisung gegeben werden. Sollen in einem Block mehrere Anweisungen verarbeitet werden,so kann man diese in einem einzigen Symbol unterbringen. Dies empfiehlt sich jedoch nur dann, wenn die Anweisungen von gleicher oder ähnlicher Art sind (z.B. Variablenzuweisungen). Trennt man in diesem Rechteck die einzelnen Anweisungen durch einen waagrechten Strich, so entsteht eine neue, zusätzliche Strukturierung.

Nassi-Shneidermann	DIN 66001
Anweisung	Anweisung

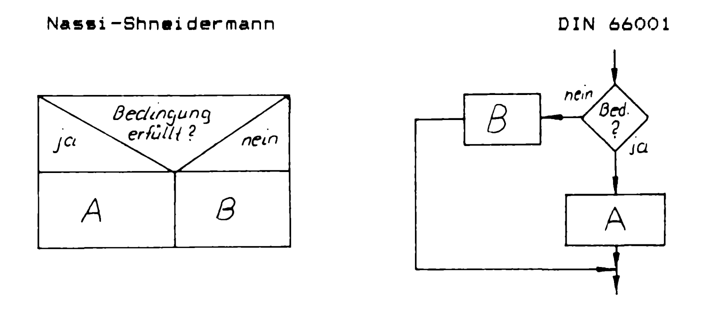
2. Der Unterprogrammaufruf

Durch dieses Symbol (siehe unten) wird kenntlich gemacht, daß ein Unterprogramm aufgerufen wird. In diesem Symbol vermerkt man in der Regel nur den Namen des entsprechenden Unterprogrammes. Die Struktur des jeweiligen Unterprogrammes braucht hier nicht näher beschrieben zu werden, da für jedes einzelne Unterprogramm ein eigenes Struktogramm erstellt werden muß.



3. Die Zweifachverzweigung

Die Zweifachverzweigung entspricht in der Regel einer IF-Abfrage, bei der ja auch - je nach Bedingung - zwei verschiedene Anweisungen ausgeführt werden. Wichtig ist, daß beide Zweige am Ende immer wieder zusammenlaufen.

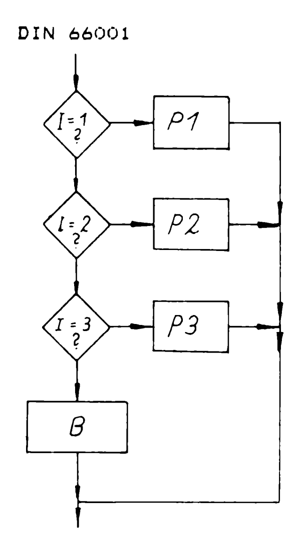


4. Die Mehrfachverzweigung

Mehrfachverzweigungen sind in der Regel dann sinnvoll, wenn man in einem Programm eine Verzweigung z. B. vom Inhalt einer numerischen Variablen abhängig macht. Üblicherweise sieht man den äußerst rechten Zweig für ungültige oder unerwartete Variablenwerte vor. Auch hier führen alle Verzweigungen am unteren Rand des Symbols wieder zusammen. Hier kann man vielleicht am besten den eigentlichen Zweck von Struktogrammen erkennen: Der Entwickler wird gezwungen, alle Programmverzweigungen nach der Abarbeitung zu einem Punkt (Stamm) zurückzuführen.

Nassi-Shneidermann

DIN 66001



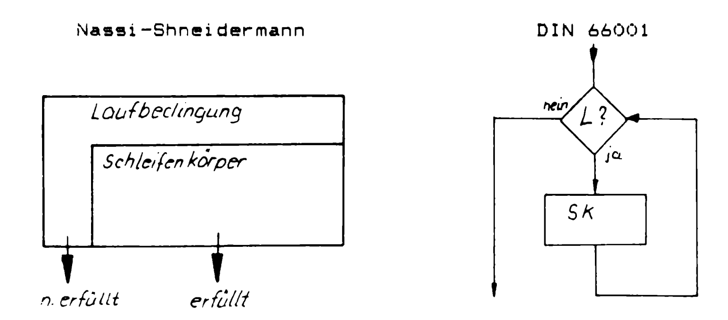


5. Die Schleife

Es gibt viele Methoden um Schleifen zu realisieren. Im Endeffekt haben jedoch alle Schleifen zwei Dinge gemeinsam: Zum einen die Laufbedingung und zum anderen eine IF-Abfrage die testet, ob die Laufbedingung (noch) erfüllt ist. Grundsätzlich kennen wir zwei Arten von Schleifen:

A) Schleife mit Vorabtest der Laufbedingung

Um die Schleife durchlaufen zu können muß zuerst die Schleifenbedingung erfüllt sein. Es ist als möglich, die Schleife komplett zu umgehen oder wie man oft sagt: 'Nullmal' zu durchlaufen. Erst wenn die Schleifenbedingung erfüllt ist, wird der Schleifenkörper durchlaufen (alle Befehle und Anweisungen, die in einer Schleife eingebettet sind, nennt man im Fachjargon 'Schleifenkörper').

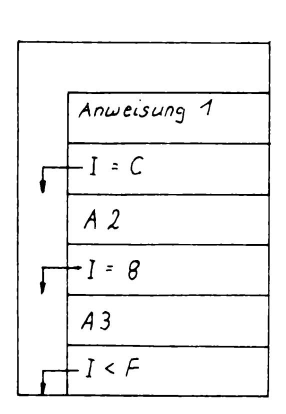


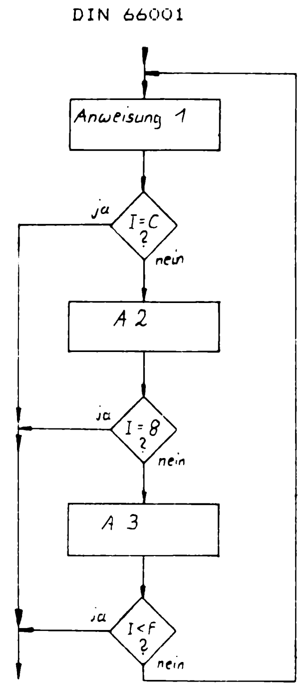
B) Schleife mit Abbruchbedingung

Hier wird in jedem Fall in den Schleifenkörper verzweigt. Innerhalb des Schleifenkörpers können mehrere Abbruchbedingungen untergebracht werden, die dazu führen, daß die betreffende Schleife an verschiedenen Stellen verlassen werden kann. Je nach Formulierung der Abbruchbedingungen kann es vorkommen, daß keine der Bedingungen zum Abbruch führt. In diesem Fall wird die Schleife neu durchlaufen.

Formulieren Sie Ihre Abbruchbedingungen immer so, daß die Schleife nach mehreren Durchläufen verlassen werden kann, sonst haben Sie einen 'Dauerläufer' im Programm, der unter Umständen eine umfangreiche Fehlersuche nach sich zieht.

Nassi-Shneidermann DIN 66001





Nun haben Sie alle Werkzeuge in der Hand, um für Ihre Programme ein Struktogramm erstellen zu können. Nach der Erstellung eines Struktogrammes für ein Programm (Unterprogramm) wird es mit einem Rechteck eingerahmt, an dessen oberem Rand der Name des Programmes vermerkt wird. Um Ihnen das eben beschriebene anschaulich zu demonstrieren, wollen wir für unser Beispielprogramm einmal ein Struktogramm erstellen (eine Darstellung des Programmes als Flußdiagramm nach DIN 66001 erübrigt sich).

Die Beschreibung der Algorhytmen für die einzelnen Programmteile entnehmen Sie bitte aus Kapitel 1.10.4.

I. Das Hauptprogramm

Hauptprogramm

Initialisierung

Graphik erzeugen

X = 1868

UPGM 11

Auf Taste warten

Q$ = CRSR UP

Q$ = CRSR Down

Q$ = CRSR Right

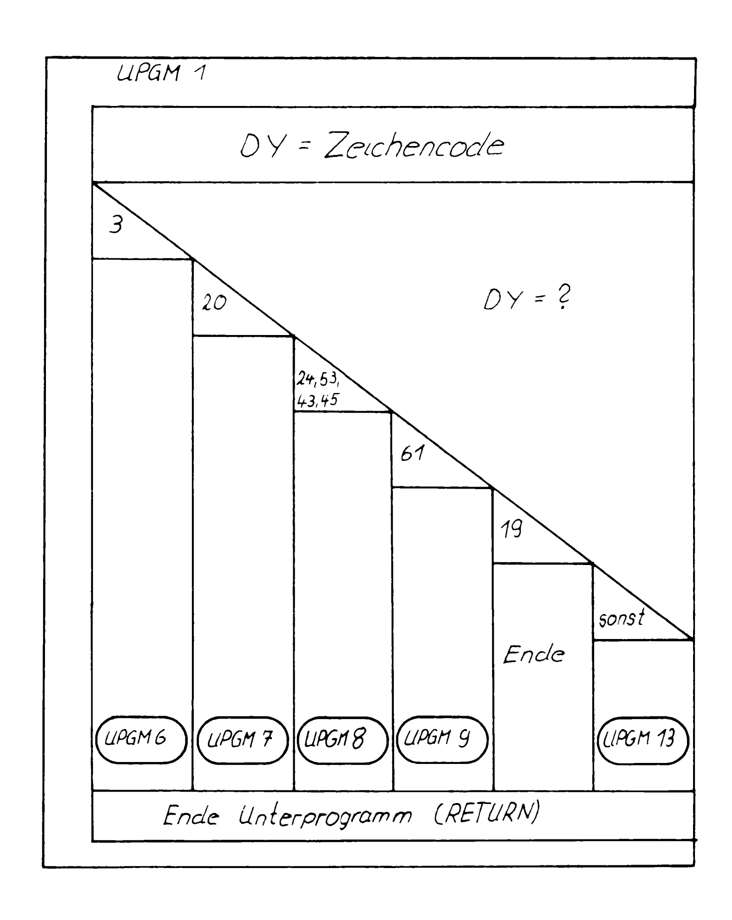
Q$ = CRSR Left

Q$ = RETURN

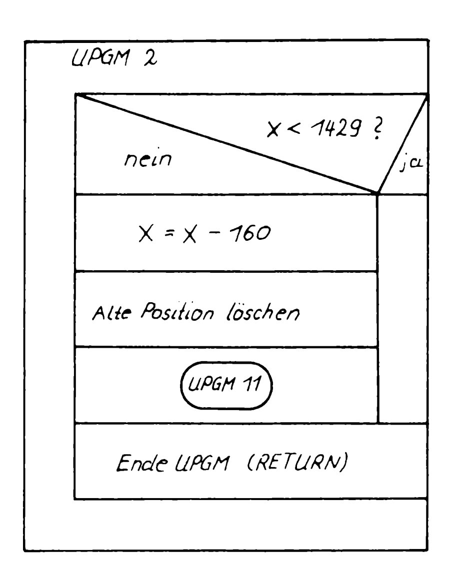
UPGM 12

Auf Taste warten

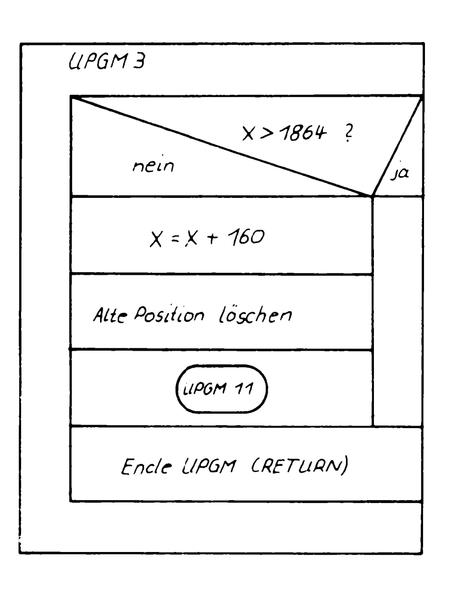
II. Unterprogramm 1: Auswertung der Eingabe



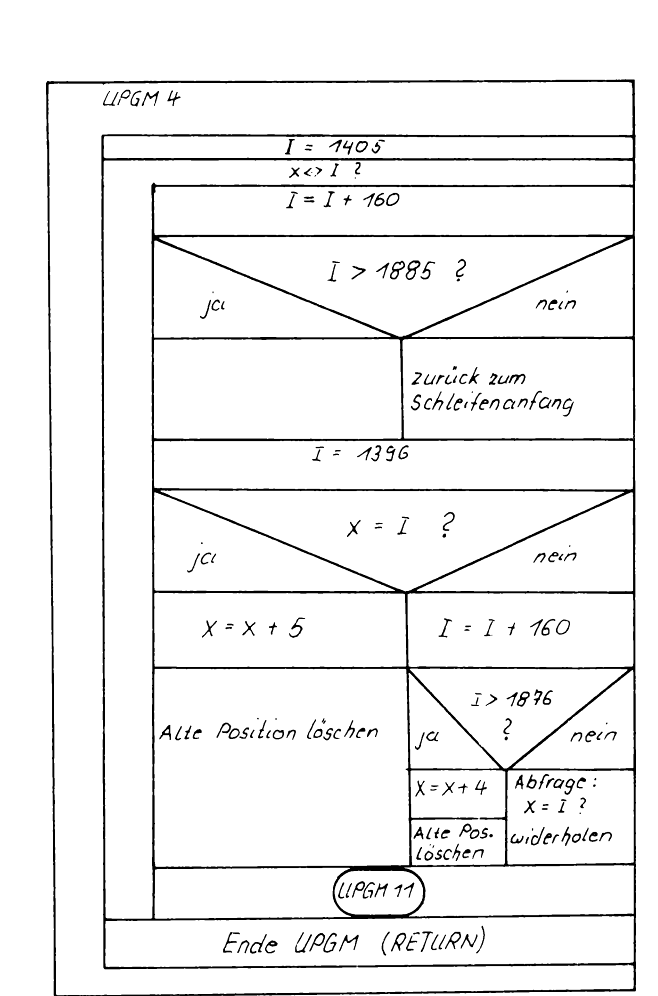
III. Unterprogramm 2: CRSR UP wurde gedrückt



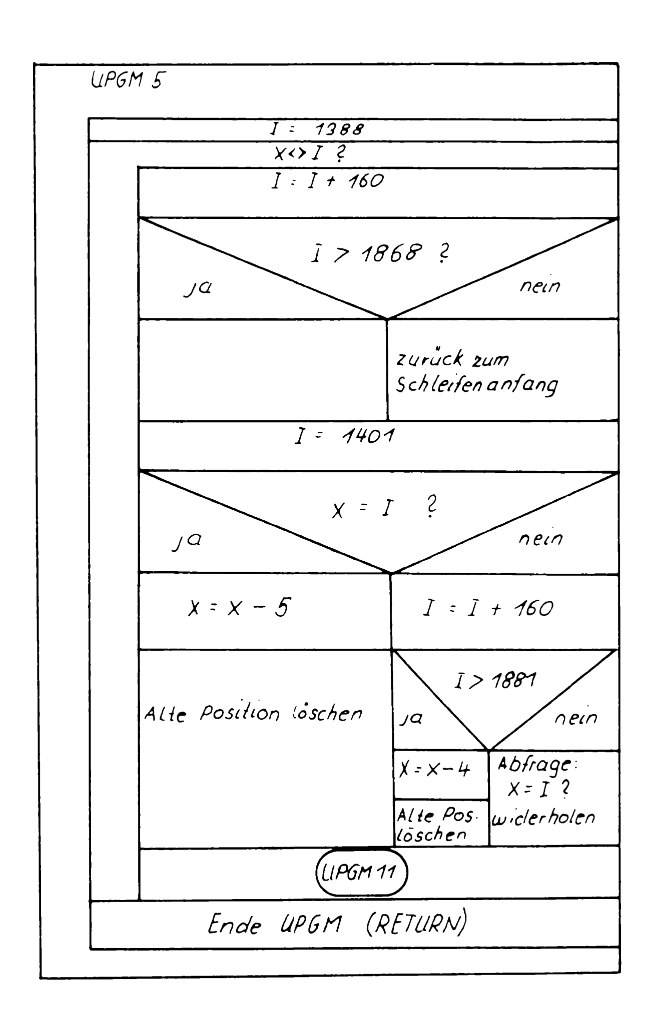
IV. Unterprogramm 3: CRSR DOWN wurde gedrückt



V. Unterprogramm 4: CRSR RIGHT wurde gedrückt



VI. Unterprogramm 5: CRSR LEFT wurde gedrückt



VII. Unterprogramm 6: Taste 'C' wurde gedrückt

UPGM 6

SU$ = ""

FLAG = 0

UPGM 10

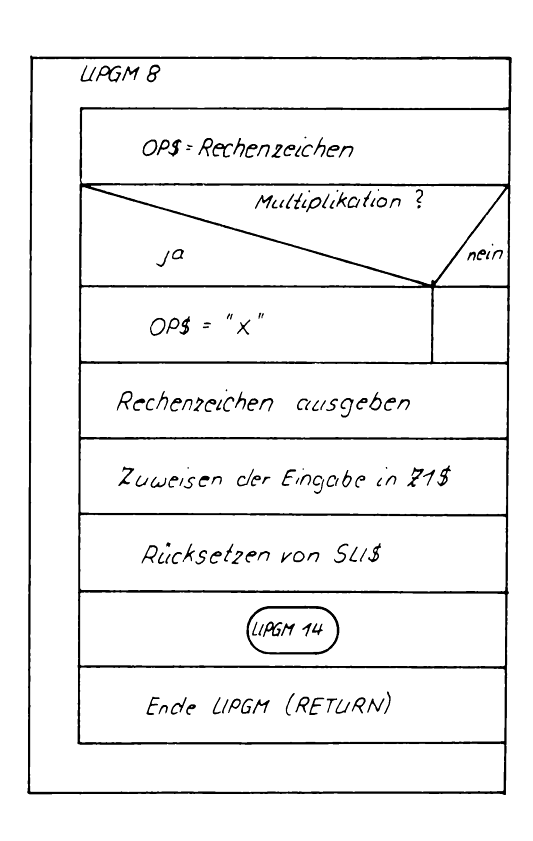
Ende UPGM (RETURN)

VIII. Unterprogramm 7: Taste 'T' wurde gedrückt

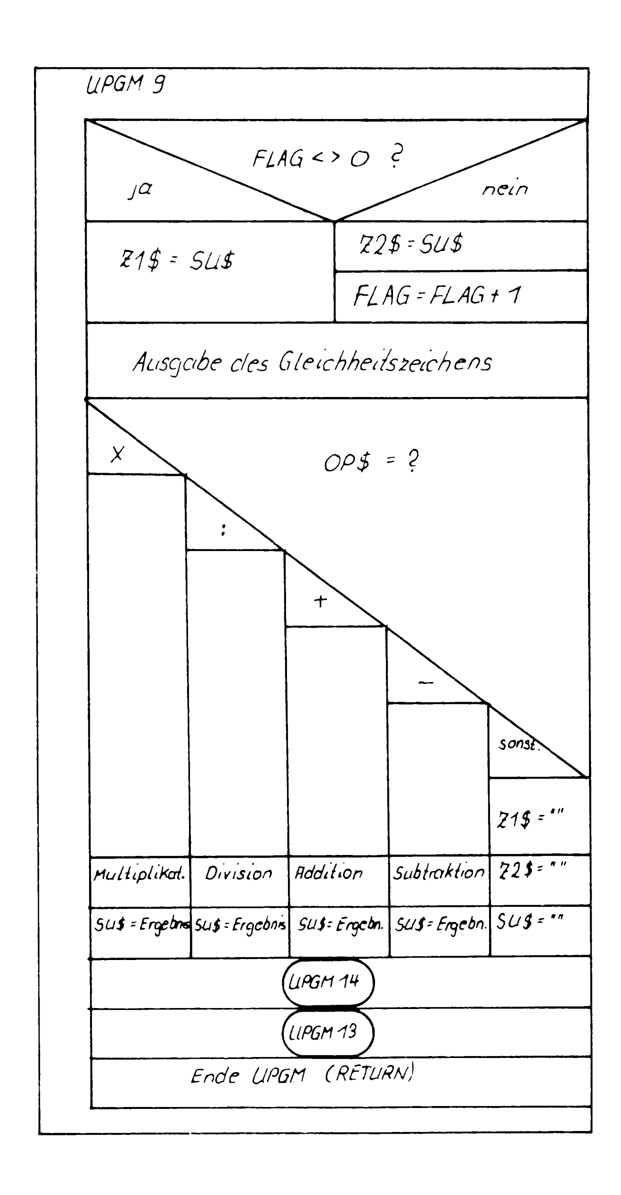
UPGM 7

Z1$ = ""
Z2$ = ''
RES = 0
FLAG = 0
(UPGM 6)
Ende UPGM (RETURN)

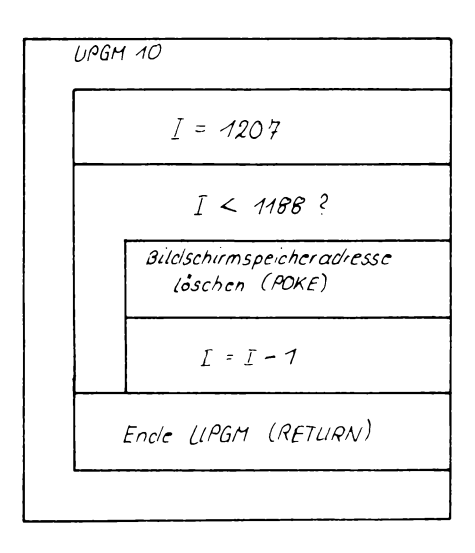
X. Unterprogramm 8: Operator wurde eingegeben



X. Unterprogramm 9: Operation ausführen



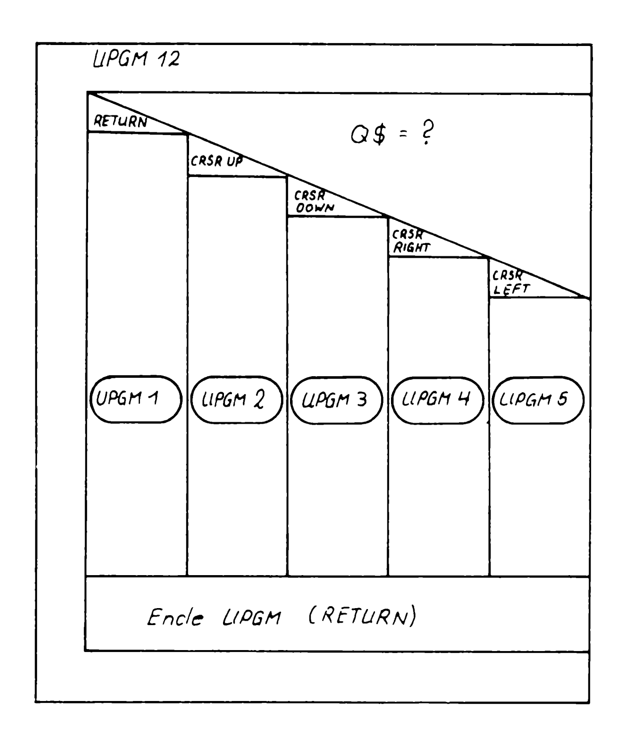
XI. Unterprogramm 10: Display löschen



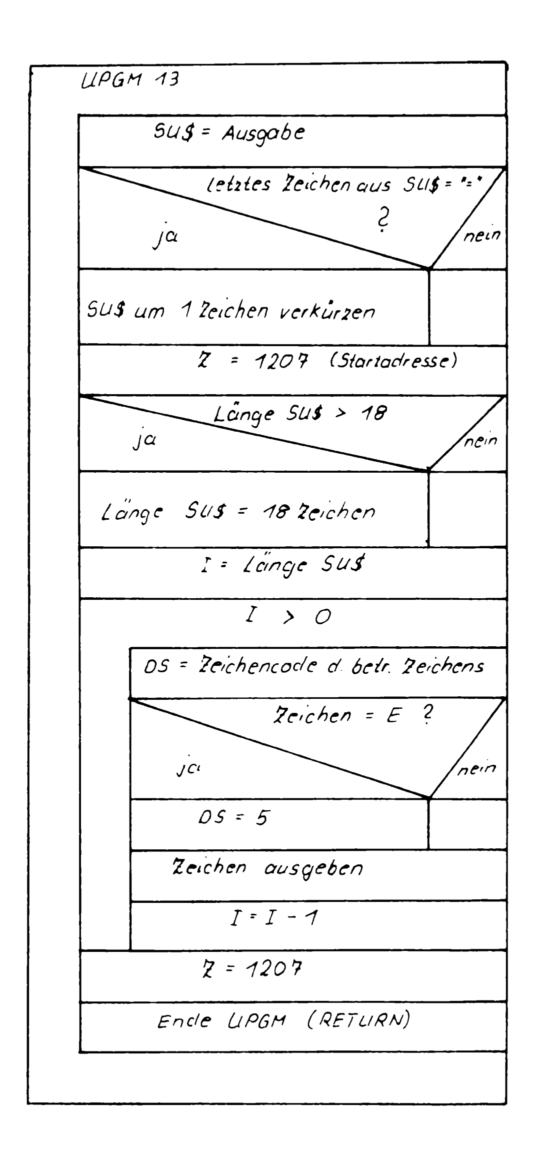
XII. Unterprogramm 11: Zeichen invers darstellen

UPGM 11
Zeichencode ermitteln und auf dem Bildschirm ausgeben: POKE X, PEEK (X) + 128
Ende UPGM (RETURN)

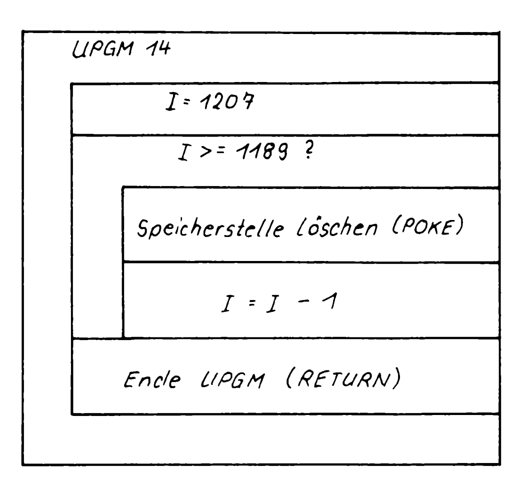
XIII. Unterprogramm 12: Welche Taste wurde gedrückt



XIV. Unterprogramm 13: Display-Ausgabe



XV. Unterprogramm 14: Nur Operator bleibt stehen



1.7. Flußdiagramm kontra Struktogramm

Sie haben jetzt beide Arten der graphischen Darstellung von Programmen kennengelernt. Sicherlich gibt es auch unter Ihnen, liebe Leser, einige Zweifler, die eine Darstellung in Form eines Flußdiagrammes nach DIN 66001 dem Struktogramm vorziehen. Bestimmt lassen sich alle Probleme auch als Flußdiagramm darstellen und übersichtlich gestalten. Auch die Analyse des Datenflußes ist auf Grund des linienförmigen Aufbaues solcher Diagramme um einiges bequemer als bei einem Struktogramm. Wir können Ihnen jedoch versichern, daß nach einer kurzen Gewöhnungsphase auch das Lesen eines Struktogrammes sehr einfach ist.

Im Folgenden wollen wir nocheinmal die programmtechnischen Vorzüge und Nachteile der beiden Darstellungsarten aufzeigen:

Das Flußdiagramm:

Kennzeichnend für diese Darstellungsart ist der einfache linienförmige Aufbau, der den Datenfluß eines Programmes oder Problemes sehr anschaulich widerspiegelt. Bei kürzeren Programmen ist es sicherlich sinnvoll, in der Programmerstellung diese Form der graphischen Darstelling zu wählen. Sollen jedoch längere und kompliziertere Programme erstellt werden ist es auch hier nötig, daß Problem in kleinere Teilprobleme zu zerlegen. Nachteilig wirkt sich hier der relativ große Platzbedarf aus, der unter Umständen zu einer gewissen Unübersichtlichkeit führen kann. Auch wird man beim Entwurf eines Flußdiagrammes nicht unbedingt dazu gezwungen ein Programm nach den Regeln der strukturierten Programmierung zu erstellen.

Das Struktogramm:

Der dominierende Vorteil dieser Darstellungsart ist die Tatsache, daß der Ersteller eines Struktogrammes im wahrsten Sinne des Wortes dazu gezwungen wird, 'strukturiert' zu denken. Durch die übersichtliche Blockgraphik ist auch hier die Erstellung sehr einfach und oft viel schneller als beim Flußdiagramm. Leider wird diese, übrigens sehr platzsparende Methode in den Medien oft vernachlässigt. Erst in letzter Zeit häufen sich Berichte und Erläuterungen zu diesen Thema. In der strukturierten Programmierung ist das - wie der Name schon sagt - Struktogramm ein unerläßliches Hilfsmittel, um Programmstrukturen leicht zu erkennen und darzustellen. Wer strukturiert programmiert oder programmieren will, muß sich mit dieser Darstellungsart früher oder später intensiv beschäftigen.

1.8. Erstellen des strukturierten BASIC-Listings

Wie an anderer Stelle schon erwähnt wurde, gibt es viele Programmierer die behaupten, daß eine strukturierte Programmierung in BASIC nicht möglich sei. Wir wollen mit Ihnen und Ihrem CBM 64 diesem Kreis beweisen, daß eine strukturierte Programmierung sehr wohl mit BASIC zu realisieren ist. Wenden wir uns zuerst einmal den Befehlen und Statements zu, die für eine strukturierte Programmierung sehr nützlich sind:

1. REM

Durch Einfügen mehrerer Kommentarzeilen ist es möglich, die verschiedenen Programmteile optisch voneinander zu trennen. Falls es der Speicherplatz erlaubt (das sollte bei den 64 K Ihres CBM 64 eigentlich immer der Fall sein !), sollte man

mit erklärenden REM-Zeilen nicht sparen, um so das Programmlisting entsprechend zu dokumentieren. Versehen Sie jedes Programmteil mit mindestens einer Kommentarzeile die darüber Aufschluß gibt, was das jeweilige Programmodul eigentlich macht.

2. GOSUB bzw. ON GOSUB

Verwenden Sie der Übersichtlichkeit halber so viele Unterprogramme wie irgend möglich. Aber nicht nur Übersichtlichkeit sondern auch Änderungsfreundlichkeit wird durch eine gute Unterprogrammtechnik gefördert, da man das betreffende Programmteil leicht und schnell auffinden und ändern kann. Vermeiden Sie nach Möglichkeit den BASIC-Befehl GOTO. GOTO ruft einen unbedingten Programmsprung hervor, der in einem strukturierten Programm nichts zu suchen hat. Versuchen Sie eine Verzweigung immer an eine Bedingung zu knüpfen, da so genau ersichtlich ist, warum der Programmfluß verzweigt. Sicherlich lässt sich der eine oder andere GOTO-Befehl nicht immer vermeiden, sollte aber erst als letzte Konsequenz benutzt werden.

Auf den folgenden Seiten haben wir unser Beispielprogramm als Listing abgedruckt. Wir hoffen, daß Sie alle Ihre Programme demnächst in dieser strukturierten Form entwickeln und auflisten werden.

>>>> UND ES GEHT DOCH IN BASIC <<<<

```
1 REM **********************
2 REM **  TASCHENRECHNER   **
3 REM **       FUER        **
4 REM **     C B M  64     **
5 REM ** COPYRIGHT (C) BY **
6 REM ** DATA BECKER GMBH **
7 REM **********************
10 REM * VARIABLEN RUECKSETZEN
20 CLR
95 REM
96 REM
97 REM
98 REM
99 REM
100 REM * ERZEUGEN DER GRAPHIK
110 PRINT"□"
120 PRINT"■■■■■■■■■ ┌──────────────────────────────┐"
130 FOR I=1TO20
140 PRINT"■■■ │                              │"
150 NEXTI
160 PRINT"■■■ └──────────────────────────────┘"
170 PRINT"■■■■■■■■ ╭─────────────────────────────╮"
180 PRINT"■■■■ │                            │"
190 PRINT"■■■■ ╰────────────────────────────╯"
200 PRINT"■■■ ▙▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▟"
210 PRINT"■■■■■■■■■■■■■■ ╭─╮  ╭─╮  ╭─╮   ╭─╮  ╭─╮"
220 PRINT"■■■■│7 │  │8 │  │9 │   │C │  │T │"
230 PRINT"■■■■╰──╯  ╰──╯  ╰──╯   ╰──╯  ╰──╯"
240 PRINT"■■■■■╭─╮  ╭─╮  ╭─╮   ╭─╮  ╭─╮"
250 PRINT"■■■■│4 │  │5 │  │6 │   │X │  │: │"
260 PRINT"■■■■╰──╯  ╰──╯  ╰──╯   ╰──╯  ╰──╯"
270 PRINT"■■■■■╭─╮  ╭─╮  ╭─╮   ╭─╮  ╭─╮"
280 PRINT"■■■■│1 │  │2 │  │3 │   │+ │  │- │"
290 PRINT"■■■■╰──╯  ╰──╯  ╰──╯   ╰──╯  ╰──╯"
300 PRINT"■■■■■╭─╮  ╭─╮  ╭─╮   ╭─╮  ╭─╮"
310 PRINT"■■■■│0 │  │. │  │E │   │= │  │S │"
320 PRINT"■■■■╰──╯  ╰──╯  ╰──╯   ╰──╯  ╰──╯"
330 PRINT"■■■";TAB(26);"■T■ = ALL CLEAR"


READY.
```

```
340 PRINTTAB(26);"[RVS ON]C[RVS OFF] = CLEAR"
350 PRINTTAB(26);"[RVS ON]E[RVS OFF] = EXPONENT"
360 PRINTTAB(26);"[RVS ON]S[RVS OFF] = ENDE"
370 PRINT"[HOME][DOWN][DOWN][DOWN][DOWN][DOWN][DOWN][RIGHT][RIGHT][RIGHT][RVS ON] DATA BECKER  CBM 64  [RVS OFF]"
395 REM
396 REM
397 REM
398 REM * BILDSCHIRMADRESSE IN X
399 REM * ABSPEICHERN UND ZEICHEN
400 REM * INVERS DARSTELLEN.
410 X=1868
420 GOSUB 10000 : REM ZEICHEN INVERS
495 REM
496 REM
497 REM
498 REM
499 REM
500 REM * TASTATURABFRAGE
510 GETQ$:IFQ$<>"[UP]"ANDQ$<>"[DOWN]"ANDQ$<>"[LEFT]"ANDQ$<>"[RIGHT]"ANDQ$<>CHR$
(13)THEN510
520 GOSUB 20000 : REM WELCHE TASTE ?
530 GOTO 510
995 REM
996 REM
997 REM
998 REM
999 REM
1000 REM * AUSWERTUNG DER EINGABE
1010 DY=(PEEK(X)-128):REM ZEICHENCODE NACH DY
1020 IF DY=3 THEN GOSUB 6000:RETURN
1030 IF DY=20THEN GOSUB 6100:RETURN
1040 IF DY=24ORDY=58ORDY=43ORDY=45THENGOSUB6200:RETURN
1050 IF DY=61THENGOSUB6300:RETURN
1060 IF DY=19THENPRINT"[CLR]":END
1070 GOSUB 20100 : REM DISPLAY-AUSGABE
1080 RETURN
1995 REM
1996 REM


READY.
```

```
1997 REM
1998 REM
1999 REM
2000 REM * CRSR UP WURDE GEDRUECKT
2010 IFX<1429THENRETURN
2020 X=X-160:POKEX+160,PEEK(X+160)-128:GOSUB10000:RETURN
2995 REM
2996 REM
2997 REM
2998 REM
2999 REM
3000 REM * CRSR DOWN WURDE GEDRUECKT
3010 IFX>1864THENRETURN
3020 X=X+160:POKEX-160,PEEK(X-160)-128:GOSUB10000:RETURN
3995 REM
3996 REM
3997 REM
3998 REM
3999 REM
4000 REM * CRSR RIGHT WURDE GEDRUECKT
4010 FORI=1405TO1885STEP40
4020 IFX=ITHENRETURN
4030 NEXTI
4040 FORI=1396TO1876STEP40
4050 IFX=ITHENX=X+5:POKEX-5,PEEK(X-5)-128:GOSUB10000:RETURN
4060 NEXTI
4070 X=X+4:POKEX-4,PEEK(X-4)-128:GOSUB10000:RETURN
4995 REM
4996 REM
4997 REM
4998 REM
4999 REM
5000 REM * CRSR LEFT WURDE GEDRUECKT
5010 FORI=1388TO1868STEP40
5020 IFX=ITHENRETURN
5030 NEXTI
5040 FORI=1401TO1881STEP40
5050 IFX=ITHENX=X-5:POKEX+5,PEEK(X+5)-128:GOSUB10000:RETURN


READY.
```

```
5060 NEXTI
5070 X=X-4:POKEX+4,PEEK(X+4)-128:GOSUB10000:RETURN
5995 REM
5996 REM
5997 REM
5998 REM
5999 REM
6000 REM * TASTE C WURDE GEDRUECKT
6010 SU$="":FLAG=0
6020 GOSUB 6900 :RETURN
6095 REM
6096 REM
6097 REM
6098 REM
6099 REM
6100 REM * TASTE T WURDE GEDRUECKT
6110 Z1$="":Z2$="":RES=0:FLAG=0:GOSUB6000 :RETURN
6195 REM
6196 REM
6197 REM
6198 REM
6199 REM
6200 REM * OPERATOR WURDE GEDRUECKT
6210 OP$=CHR$(DY):IFASC(OP$)=24THENOP$="X"
6220 PRINT"SQQQQ]]]]]";OP$
6230 Z1$=SU$:SU$="":GOSUB20500
6240 RETURN
6295 REM
6296 REM
6297 REM
6298 REM
6299 REM
6300 REM * OPERATION AUSFUEHREN
6310 IFFLAG<>0THENZ1$=SU$:GOTO6330
6320 Z2$=SU$:FLAG=FLAG+1
6330 PRINT"SQQQQ]]]]]";"="
6340 IFOP$="X"THENRES=VAL(Z1$)*VAL(Z2$):GOSUB20500 :SU$=STR$
(RES):GOSUB20100:RETURN


READY.
```

```
6350 IFOP$=":"THENRES=VAL(Z1$)/VAL(Z2$):GOSUB20500 :SU$=STR$
(RES):GOSUB20100:RETURN
6360 IFOP$="+"THENRES=VAL(Z1$)+VAL(Z2$):GOSUB20500 :SU$=STR$
(RES):GOSUB20100:RETURN
6370 IFOP$="-"THENRES=VAL(Z1$)-VAL(Z2$):GOSUB20500 :SU$=STR$
(RES):GOSUB20100:RETURN
6380 Z1$="":Z2$="":SU$="":GOSUB20500:GOSUB20100:RETURN
6895 REM
6896 REM
6897 REM
6898 REM
6899 REM
6900 REM * DISPLAY LOESCHEN
6910 FORI=1207TO1188STEP-1
6920 POKEI,32
6930 NEXTI
6940 RETURN
9995 REM
9996 REM
9997 REM
9998 REM
9999 REM
10000 REM * ZEICHEN INVERS DARSTELLEN
10010 POKEX,PEEK(X)+128
10020 RETURN
19995 REM
19996 REM
19997 REM
19998 REM
19999 REM
20000 REM * WELCHE TASTE WURDE GEDRUECKT
20010 IF Q$=CHR$(13)THENGOSUB1000:RETURN
20020 IF Q$="⏋"THENGOSUB2000:RETURN
20030 IF Q$="▣"THENGOSUB3000:RETURN
20040 IF Q$="▮▮"THENGOSUB4000:RETURN
20050 IF Q$="▮▮"THENGOSUB5000:RETURN
20095 REM
20096 REM


READY.
```

```
20097 REM
20098 REM
20099 REM
20100 REM * DISPLAY-AUSGABE
20110 SU$=SU$+CHR$(DY):IFRIGHT$(SU$,1)="="THENSU$=LEFT$(SU$,
LEN(SU$)-1)
20120 Z=1207
20130 IFLEN(SU$)>18THENSU$=RIGHT$(SU$,18)
20140 FOR I=LEN(SU$)TO1STEP-1
20150 DS=ASC(MID$(SU$,I,1))
20160 IFMID$(SU$,I,1)="E"THENDS=5
20170 POKEZ,DS
20180 Z=Z-1
20190 NEXTI
20200 Z=1207
20210 RETURN
20495 REM
20496 REM
20497 REM
20498 REM
20499 REM
20500 REM * NUR OPERATOR BLEIBT STEHEN
20510 FORI=1207TO1189STEP-1
20520 POKEI,32
20530 NEXTI
20540 RETURN

READY.
```

2 EIN PROGRAMM BESTEHT AUS DREI DINGEN

Sollte die überschrift zu diesem Teil des Buches in Ihnen Assoziationen zu einer bestimmten Tabakwerbung wecken, gemäß derer ein Mann drei Dinge brauche, nämlich Feuer, Pfeife und ......, so haben wir das keineswegs beabsichtigt.
Jedoch während das Obengenannte entbehrlich ist, sind es die Dinge, von denen hier die Rede sein soll, für ein vernünftiges Programm keineswegs, sondern sind notwendiger Bestandteil desselben.
In einem Programm, sofern es nicht aus lauter REM's oder der Zeile

```
10 goto 10
```

besteht, stellen sich immer die Fragen:

WAS TUN?　　　　WOMIT?　　　　WOHIN?

Selbst die einfache Zeile

```
10 print "hallo"
```

rechnet mit diesen drei Kriterien. WAS TUN?... nämlich Drucken, WOMIT?... das in der Zeile selbst enthaltene Literal 'hallo', WOHIN?.... auf den Bildschirm.
Die Frage WOMIT beantwortet sich der ernsthafte Programmierer natürlich nicht mit im Programm selbst untergebrachten Literalen, soweit es sich nicht um für Formeln benötigte Konstanten handelt, sondern diese Daten kommen in aller Regel, zumindest bei der Ersterfassung, von der Tastatur.
Folgende drei Dinge sind also Gegenstand der Betrachtungen:

WOMIT?	Dateneingabe
WAS TUN?	Datenverarbeitung
WOHIN?	Datenausgabe

Ohne dabei näher auf anwendungsspezifische Problematiken einzugehen, wollen wir Ihnen anhand dieser drei Oberbegriffe grundsätzliche Methoden vorstellen, die Sie in die Lage versetzen sollen, leichter Programme zu erstellen, die

professionellen Anforderungen genügen. Darunter verstehen wir Programme, die nicht nur der Ersteller bedienen kann, sondern auch (und gerade) der Anwender.
Zu allen Punkten bieten wir Ihnen auch Subroutinen an, deren Zeilennumerierung und Variablenwahl so erfolgt, daß Sie diese alle miteinander in einem Programm unterbringen können.
Soweit es möglich ist, sind alle Routinen parametrisiert, d.h. daß Sie diese durch einfache Veränderung von Variableninhalten den Erfordernissen Ihres Programmes entsprechend anpassen können.
Durch das gesamte Kapitel 2 wird Sie, um den Stoff zu veranschaulichen, als Beispiel eine universelle Adressdatei mit dynamischer Länge der Einträge und dynamischer Freispeicherverwaltung begleiten.
Alles, was Sie dazu brauchen, ist am Ende nur noch die sinnvolle Aneinanderreihung der beschriebenen Unterprogramme mit entsprechender Parameterübergabe.
Ob Sie dann damit Adressen oder Ihre Schallplattensammlung oder ....
sehen Sie selbst.

2.1 DATENEINGABE

Was man auch darunter verstehen kann, soll Ihnen ein (wahres) Beispiel aus der täglichen Praxis des Autors als Leiter der DATA BECKER Entwicklungsabteilung zeigen:
Zur Klärung bestimmter Probleme suchte uns ein Kunde auf. Auf daß ihm rasch geholfen werde, hatte er gleich seine Programme mitgebracht. Hierbei handelte es sich um die statistische Auswertung bestimmter Daten.
Wir staunten nicht schlecht, als sich die gut fünfzehn Programme von der Logik her als identisch herausstellten. Was sie unterschied, waren einzig und allein die Argumente zu den Berechnungen. Diese waren samt und sonders als Literale in den Programmen selbst verankert.
Ja, werden Sie sagen, er hätte doch nur den Programmrumpf mit ein paar einfachen Inputs versehen brauchen und die

Ergebnisse zwecks beliebig häufigem Ausdruck auf Diskette auslagern brauchen, und außerdem ....
Genau darum geht es in diesem Buch und wir beginnen mit der Dateneingabe über Tastatur.

2.1.1 DIE BILDSCHIRMMASKE

Unter einer Maske soll in diesem Zusammenhang ein fester Bildschirmaufbau verstanden werden oder, wenn Sie so wollen, auch eine Schablone, die mit fixen Feldern versehen ist, die (und nur die) ausgefüllt werden können wie bei einem Formular
Eine gute Bildschirmmaske ist die Visitenkarte eines Programmierers, da dies gewöhnlich das Einzige ist, was der Anwender vom Programm zu sehen bekommt.

Eine Maske dient dazu, eine fehlerfreie Eingabe zu unterstützen. Der Benutzer soll jederzeit genau wissen, welche Aktion von Ihm verlangt wird.
Deshalb ist es äußerst wichtig und zweckmäßig, daß eine Maske für einen bstimmten Programmzyklus immer die gleiche ist, damit jede Unsicherheit möglichst ausgeschlossen wird.

Die Maske setzt sich am besten aus drei Hauptfeldern zusammen, nämlich:

1. Einer Kopfzeile, in der angezeigt wird, in welchem Programm(teil) man sich gerade befindet und gegebenenfalls auch der Name der augenblicklich bearbeiteten Datei steht.

2. Einem Eingabefeld, welches in mehrere Einzelfelder, je nach logischem Zusammenhang, unterteilt sein kann und aus welchen der Cursor nur nach formal richtiger Eingabe fortbewegt werden kann.

3. Einer Fußzeile, aus der ersichtlich ist, auf welche Weise das Programmteil verlassen werden kann, wenn nicht

die Art der Eingabe die programmtechnische Entscheidung ermöglicht, d.h. die Eingabe nicht einmalig ist wie z.B. das Tagesdatum.

Befassen wir uns zunächst mit einem grundsätzlichen Thema, nämlich dem farblichen Aufbau der Maske.
Der Commodore 64 bietet hier reizvolle Möglichkeiten des Maskenaufbaues. Bei Verwendung eines Farbfernsehers bieten sich natürlich die verschiedenen Farben zur optischen Aufteilung des Bildschirmes an. Jedoch ist hier Vorsicht geboten, daß man nicht des Guten zuviel tut. Es sollten auf jeden Fall die Regeln der Ergonomie beachtet werden, damit das Auge nicht irritiert oder durch ungeschickte Farbwahl ermüdet wird. Eine Bildschirmmaske ist kein Kunstwerk, mit dem der Programmierer Eindruck schindet, sondern Arbeitsmittel für den späteren Anwender, mit dem dieser unter Umständen tagtäglich umgehen muß. Denken Sie bitte unbedingt daran bei der farblichen Gestaltung Ihrer Maske. Effekthascherei und der kräftige Griff in den Farbtopf erweisen sich meist als Bumerang.

Aber auch bei Verwendung eines grün/schwarz-Monitors rufen die Farben unterschiedliche Helligkeitseindrücke hervor. Hier haben Sie sogar die Möglichkeit, es den teuren Terminals nachzutun, die ja auch verschiedene Helligkeitsstufen haben. Wir haben es ausprobiert und herausgefunden, daß der optimale Eindruck bei einem Monitor entsteht, wenn Hintergrund und Rahmen schwarz gesetzt werden, feststehende Informationen (die Maske) in grau 2 und die tatsächlichen Eingaben in weiß. Damit kann man die bei monochromen Rechnern weit verbreitete Unsitte umgehen, die unveränderlichen Bildteile in reverser Darstellung (dunkle Schrift auf hellem Grund) anzuzeigen, was erstens schlechter lesbar ist und zweitens die Augen sehr ermüdet, da der Kontrast zur normalen Schrift sehr groß ist. Reverse, und damit auch auffällig, sollten nur Fehlerzustände signalisiert werden, z.B. der Fehlerkanal der Floppy.

Viele von Ihnen werden jetzt vielleicht sagen, das könne doch wohl nicht stimmen. Gewiß, es gibt viele sogenannte professionelle Programme, die vor REVERSE-Darstellungen nur so strotzen. Betrachten wir aber einmal Programme, die typischerweise die Augen des Benutzers stark beanspruchen, nämlich Textverarbeitungsprogramme. Hier werden Sie REVERSE-Darstellungen im Normalbetrieb so gut wie nie finden.

Von Ergonomen immer mehr gefordert wird aber die komplette REVERSE Darstellung des Bildschirmes, d.h. grundsätzlich sollen der gesamte Hintergrund hell und die Schrift dunkel sein. Dazu ist zu sagen, daß hierzu besondere Monitore erforderlich sind, die tatsächlich die Brillanz einer Schreibmaschinenseite erreichen. Der Effekt ist ähnlich, wie er Ihnen vielleicht von Perlleinwänden für Projektoren bekannt ist. Ein normales Fernsehgerät kann jedoch diese Auflösung und Reinheit des Bildes nie erreichen.

Wir wollen uns auf das beim C64 Machbare beschränken. Hier unsere Farbempfehlung und die dazugehörenden Befehle, die am Programmanfang stehen sollten:

```
poke 53280,0:rem rand schwarz
poke 53281,0:rem hintergrund schwarz
printchr$(152);:rem einleiten schrift grau 2
printchr$(5);:rem einleiten schrift weiss
```

wobei die weisse Schrift zur Anzeige der Aktivitäten dienen soll, und grau 2 für konstante Teile wie die Maske, was wir im nächsten Abschnitt mit einem Maskengenerator konkretisiert haben.

Lasse Sie uns noch einmal zusammenfassen: Wichtig ist nicht, daß ein unbedarfter Beobachter ob der Farbenpracht auf dem Bildschirm in Verzückung gerät, sondern vielmehr, daß der spätere Benutzer damit möglichst gut arbeiten kann.

2.1.2 EINGABEFELDER

Bei dem von uns vorgeschlagenen Maskenaufbau und der gegebenen Bildschirmgröße von 25 Zeilen zu je 40 Zeichen ergeben sich eine Kopfzeile, 23 Eingabezeilen und eine Fußzeile.
Um nun zu unserem Beispiel der Adressdatei zu kommen, besteht wohl die erste Aufgabe darin, die Stammdaten einzugeben, d.h. Name, Anschrift usw.

Die Kopfzeile könnte dann enthalten:
STAMMDATENERFASSUNG   DATEINAME

Entsprechend muß auch das Feld aufgebaut sein, und zwar, und das gilt für jede Art von Eingabe, in der logischen Reihenfolge, das heißt in der Form, in der der Benutzer die Daten gewöhnlich auch vorliegen hat.
Bei uns hieße das:

Anrede
Name
Straße
Hausnummer
Postleitzahl
Ort
Bemerkung 1
:
:
Bemerkung n

Die Fußzeile könnte lauten:
EINGABEENDE: Funktionstaste F1 im Feld 'Anrede'

Wir haben zur Meldung des Eingabeendes an das Programm hier eine Funktionstaste vorgesehen. COMMODORE 64-Besitzer sind in der glücklichen Lage, über vier (ohne SHIFT) dieser in der professionellen EDV unentbehrlichen Funktionstasten verfügen zu können. Die Wichtigkeit von Funktionstasten für

die Bedienerführung in einer Eingabemaske ist leicht einzusehen:
Drei Dinge (schon wieder drei) werden regelmäßig in jeder Eingabemaske auftauchen, nämlich die Abfragen:

1. ALLE DATEN RICHTIG - SOLL ABGESPEICHERT WERDEN?
2. EINGABEENDE (oft gleichbedeutend mit PROGRAMMENDE für diesen Programmteil und in der Regel Rückkehr ins Hauptmenü).
3. die aus Komfortgründen <u>stets anzubietende</u> Möglichkeit, jeweils ein Eingabefeld zurückzuspringen.

Alle diese Abfragen sollten grundsätzlich nicht die Eingabe von mehr oder weniger originellen Dingen erfordern (z.B. # für Programmende), sondern lediglich das Drücken einer Funktionstaste. Dabei sollte die der einzelnen Funktion zugeteilte Taste nicht nur für alle Programmteile, sondern für möglichst alle Ihre Programme gleich sein. Dies trägt ebenso zur bedienerfreundlichen Gestaltung eines Programmes bei, wie ein einheitlicher Maskenaufbau.
Eine entsprechende Anweisung sähe (unter Zugrundelegung der Inputroutine aus 2.1.4 sähe für z.B. das Programmende so aus:

```
100f%=1:gosub49000:if f$=chr$(133) then end
```

Doch zunächst zurück zum Maskenaufbau:
Mit Hilfe der am Ende dieses Abschnitts gelisteten und kommentierten Subroutine macht Ihnen der Maskenaufbau sicherlich keine Mühe.
Vor dem Aufruf sind folgende Variable zu versorgen:

f% enthält die gewünschte Feldnummer und kann Werte von 0 bis 24 annehmen, wobei 0 die Kopfzeile darstellt und 24 die Fußzeile.

f$ soll den festen Bestandteil des Feldes enthalten, also z.B. 'Anrede'.

fl% gibt die Feldlänge an, worunter die Länge des tatsächlichen Eingabeteiles verstanden werden soll. MAXIMUM: len(f$)+fl% darf 37 nicht überschreiten. fl% darf auch 0 sein, was bedeutet, daß dieses Feld nur der

Anzeige dient und keine Eingabe verlangt wird.

ff% bestimmt die Feldart:

=0: Eingabe alphanumerisch, Länge darf kleiner als fl% sein

=1: Eingabe alphanumerisch, Länge muß fl% entsprechen

=2: Eingabe numerisch, Länge darf kleiner als fl% sein

=3: Eingabe numerisch, Länge muß fl% entsprechen

Beispiel:

```
10 f%=1:fl%=5:ff%=0:f$="Anrede":gosub 51000
```

Damit sind schon zwei wesentliche Dinge erledigt, nämlich der Bildaufbau und, was allerdings erst bei der Dateneingabe zum Tragen kommt, der Feldtyp.
Das nächste wäre eine evtl. Übergabe von Defaultwerten, d.h. es werden in die Eingabefelder bereits Werte eingetragen, die wahrscheinlich ohnehin eingegeben würden. Der Regelfall ist eine solche Vorbesetzung bei der Änderung von Stammdaten, die vielleicht notwendig werden, weil sich die Anschrift oder die Telefonnummer geändert hat, der übrige Eintrag aber erhalten bleiben soll.
In dem Falle werden die Eingabefelder einfach mit den von der Diskette gelesenen Daten gefüllt. Die nicht zu ändernden Felder werden vom Benutzer einfach mit der Taste 'RETURN' quittiert.
Auch kann der Fall auftreten, daß Sie es bei Ihren Adressen größtenteils mit Damen zu tun haben. Sie könnten dann das Feld 'Anrede' schon mit 'Frau' vorbesetzen. Bei der ersten Eingabe in dieses Feld (falls es sich doch um einen Herrn handelt) wird die Vorbesetzung gelöscht.
Folgende Parameter sind zu übergeben:

f% Feldnummer

f$ die Daten, mit denen Sie gerne das Eingabefeld vorbesetzt hätten.
Achtung: f$ wird auf die Feldlänge begrenzt.

Beispiel:

```
20 f%=1:f$="Frau":gosub 50000
```

Die Entscheidung, wann eine bestimmte Maske zu verlassen ist, kann sowohl vom Anwender als auch vom Programm getroffen werden.
Das Programm bricht die Maske regelmäßig dann ab, wenn die Eingabe von ihrer Natur her einmalig ist, z.B. Auswahl eines neuen Programmes aus einem Menü.
Bei repetierenden Eingaben, eben unserer Stammdatenerfassung, muß der Benutzer die Maske beenden. Das geschieht in solchen Fällen am besten durch Eingabe einer Funktionstaste.
Um dem Anwender diese Möglichkeit zu signalisieren, dient die Fußzeile.

Beispiel:

```
15 f%=24:f$="Programmende: F1 im Feld Anrede eingeben":gosub
   51000
```

Zu dem nachfolgenden Programmlisting noch eine kleine Anmerkung:
Ab der Zeile 51500 ist noch eine kleine Routine untergebracht, die es Ihnen ermöglicht, einzelne oder alle Eingabefelder zu löschen. Kopf- und Fußzeile bleiben davon unberührt. Das globale Löschen ist nach dem Ausfüllen und abspeichern einer kompletten Maske vor einer erneuten Eingabe zweckmäßig. Das gezielte Löschen könnte erforderlich werden, wenn Sie bei einer der Eingabe folgenden Plausibilitätskontrolle feststellen, daß etwas nicht stimmt, z.B. ein Datum.
Zum Löschen ist nur die Feldnummer in f% zu übergeben. Ist f%=0, werden alle Eingabefelder gelöscht außer Kopf- und Fußzeile. Diese können aber im Bedarfsfalle auch überschrieben werden.

```
50000 rem
50001 rem************** defaulteingabe **************
50002 rem
50020 iff%<1orf%>23thenreturn
50040 poke214,f%:poke211,f%(f%,2):sys58732
50060 printchr$(5)left$(f$,f%(f%,0));
50080 f%(f%,3)=1:return
51000 rem
51001 rem************** maskenaufbau ************
51002 rem
51005 poke53280,0:poke53281,0
51010 iff%<0orf%>24thenreturn
51020 cb$="                                        "
51030 ondf%goto51060
51040 dimf%(24,3):df%=1
51060 iff%=0thenprintchr$(19)chr$(152)left$(f$,40):return
51070 iff%<>24then51100
51080 poke214,24:poke211,0:sys58732
51090 printchr$(152)left$(f$,39);:return
51100 f%(f%,0)=fl%:f%(f%,1)=ff%
51120 poke214,f%:poke211,0:sys58732
51130 printchr$(152)f$;
51135 iffl%=0thenreturn
51137 print" "chr$(181);
51140 f%(f%,2)=peek(211)
51150 printleft$(cb$,fl%)chr$(182);
51160 f%(f%,3)=0
51180 return
51500 rem
51501 rem**************** feld loeschen **************
51502 rem
51510 iff%<0orf%>24thenreturn
51520 iff%<>0then51600
51540 forl1=1to23
51560 f%=l1:gosub51600
51580 nextl1:return
51600 poke214,f%:poke211,f%(f%,2):sys58732
51620 printleft$(cb$,f%(f%,0));
51640 f%(f%,3)=0:return
```

Die Variablen haben wir Ihnen oben bereits vorgestellt.
Sie finden im Listing häufig das folgende Kommando:
sys58732
Dieser Befehl veranlaßt das Betriebssystem, den Cursor auch tatsächlich an die in den Speicherstellen 211 und 214 hinterlegte Position zu bringen.
Nun zum Kommentar. Wegen der Durchsichtigkeit des Programmes fällt dieser recht knapp aus.

50000-
50080 Zwecks Ausfüllen eines Eingabefeldes mit einem Defaultwert wird der Cursor auf die in f%(f%,2) hinterlegte Position des Eingabefeldes gesetzt. Der Defaultwert in f$ wird nun maximal mit der Länge des Eingabefeldes dorthin geschrieben.Zum Zeichen, daß dieses Feld nun belegt ist, wird ein entsprechender Merker in f%(f%,3) gesetzt. Dieser wird in der Input-Routine (siehe dort) abgefragt.

51000-
51180 Wo das Eingabefeld beginnt, wird durch die Position des Cursor nach Printen des Maskentextes und anschließendem Setzen der linken Feldbegrenzung (chr$(181)) aus der Speicherstelle 211 ermittelt und in f%(f%,2) hinterlegt.
In der Länge des Eingabefeldes werden nun Blanks geschrieben, daran anschließend die rechte Feldbegrenzung. Der Merker für die Feldbesetzung wird =0 gesetzt.

51500-
51640 Das Löschen des oder der Felder geschieht dadurch, daß aus f%(f%,x) die Information über das Feld entnommen wird, der Cursor auf den Feldanfang gesetzt und in der Länge des Feldes dieses mit Leerzeichen überschrieben wird.

Das war schon alles. Wie Sie bequem die Feldinhalte in Ihr Programm übernehmen können, zeigt Ihnen der nächste

Abschnitt.

2.1.3 CURSORPOSITIONIERUNG

Unabhängig von der im Abschnitt 2.1.4 zu besprechenden Eingabemethode stellt sich grundsätzlich die Frage, wie dem Benutzer am augenfälligsten klargemacht wird, was er tun soll.

Dazu zählt neben einem logischen Aufbau der Maske auch die Hinführung der Augen auf die Stelle, an der die Eingabe erscheinen wird. Dem Empfinden nach wird, wie beim Lesen, der Aufbau von oben nach unten und von links nach rechts erwartet.

Diesem Umstand sollten Sie Rechnung tragen und die Eingabe nicht durch wildes Springen von einem Feld über das nächste und wieder zurück erschweren.

In unserer Routine werden die Eingabefelder sowohl am Anfang als auch am Ende durch Balken markiert. Dadurch ergibt sich rein optisch schon die Länge, sodaß niemand auf den Gedanken kommen könnte, in dem Feld für die Anrede den Straßennamen unterzubringen.

Zur endgültigen Eingabeaufforderung erscheint in dem zu behandelnden Feld der blinkende Cursor

Jetzt beginnt der schwierigste Teil der Unternehmung. Mit Rücksicht auf die jeweils verlangten Daten, nicht zuletzt im Hinblick auf den dafür reservierten Platz auf der Diskette, darf es einfach nicht vorkommen, daß mehr Daten eingegeben werden, als durch die Feldmarkierung vorgegeben ist. Darüberhinaus darf der Cursor das Feld nach keiner Richtung verlassen können, was die ganze Eingabe durcheinanderbringen würde.

Also muß die Cursorbewegung laufend überwacht werden. Leider haben wir beim C64 nicht die Möglichkeit einer Fensterdefinition, wie sie vom CBM8032 her bekannt ist. Die Eingabestatements INPUT und GET bieten hier auch keinen Ansatzpunkt.

Es bleibt also nichts anderes übrig, als nach jedem Tastendruck die Cursorpositon sowohl horizontal mit Feldanfang und -ende zu vergleichen, als auch vertikal ein

evtl. Verlassen der Zeile zu prüfen.
Dazu gibt es in der Zeropage (das ist der Adressraum 0-255, in dem das Betriebssystem Ablaufparameter hinterlegt) zwei Speicherstellen, die die augenblickliche Cursorposition wiedergeben.
Das ist zum einen die Adresse 211, in der die Cursorspalte, also die Position des Cursor innerhalb einer Zeile nachzusehen ist, und zum anderen die Adresse 214, deren Inhalt die aktuelle Cursorzeile reflektiert.
Diese laufende Überwachung ist natürlich nur bei Eingaben mittels GET möglich, da nur hierbei das Programm nach jedem Tastendruck wieder die Kontrolle erhält, wie die INPUT-Routine im Abschnitt 2.1.7 zeigt.

Eine gänzlich andere Situation entsteht, wenn dem Benutzer die Wahl gelassen werden soll. welches Feld er ausfüllen darf.
In dieser Form tritt es fast regelmäßig bei einer menümäßigen Programm- oder Aktionsauswahl auf.
In solchen Fällen hat sich bisher immer die Ziffern- oder Kennbuchstabenauswahl bewährt. Auch hierfür ist unser Maskengenerator gerüstet. Im vorigen Abschnitt mochte es wohl einigen unverständlich erscheinen, wozu man wohl ein Feld der Länge 0 definieren sollte. Hier bekommt es einen Sinn.
Angenommen, wir wollen drei Programme zur Auswahl stellen, wovon eines mittels Kennziffer zu bestimmen ist. Die Maske kann dann folgendermaßen aufgebaut werden:

```
10 f%=0:f$="Hauptmenue":gosub 51000
20 f%=2:fl%=1:ff%=3:f$="Stamm anlegen      = 1":gosub 51000
30 f%=4:fl%=0:f$="Stamm ändern        = 2":gosub 51000
40 f%=6:f$="Nach Namen suchen = 3":gosub 51000
50 f%=24:f$="Bitte  Ziffer  des  Programmes  eingeben":gosub
   51000
```

Die Eingabe erfolgt nur im Feld 2.
Auf diese Weise kann auch in ähnlichen Fällen verfahren werden, ohne den Benutzer durch eine plötzlich geänderte

Syntax zu verunsichern.
überhaupt sollte man sich es angewöhnen, immer nur gleichartige Aktionen zu verlangen. Oft wird nämlich der Fehler gemacht, daß längere Eingaben mit INPUT geholt werden, was beim Anwender nach der Eingabe das anschließende Drücken der Taste RETURN erfordert, wohingegen an anderer Stelle im gleichen Programm eine einfache ja/nein-Entscheidung mittels GET eingelesen wird, wonach RETURN nicht erforderlich ist und dem Bediener die Möglichkeit einer Korrektur genommen wird.
Es sollte grundsätzlich die Bestätigung mit RETURN verlangt werden, da bis dorthin die Eingabedaten noch korrigiert werden können, was aber dennoch nicht die problematische Eingabe mit INPUT notwendig macht.
Eine Ausnahme bilden die Funktionstasten. Diese sollten als Steuertasten, wie RETURN, unmittelbar eine Aktion veranlassen können. Steuertasten brauchen nicht quittiert zu werden (RETURN selbst wird ja auch nicht besonders bestätigt).
Mehr darüber und eine entsprechende Routine jedoch im nächsten Abschnitt.

2.1.4 TASTATURüBERNAHME

Hinter diesem Titel verbirgt sich dreierlei:

1. Die Abfrage der Tastatur
2. Die simultane oder nachträgliche Anzeige am Bildschirm
3. Die fliegende oder abschließende Plausibilitätskontrolle

Zur Abfrage der Tastatur bieten sich im wesentlichen zwei Verfahren an, die beide sowohl Vor- als auch Nachteile besitzen.
INPUT
Dieses Kommando besitzt den Vorzug, daß es den blinkenden Cursor auf dem Bildschirm erscheinen läßt, was vom Bediener eigentlich immer als Aufforderung zur Eingabe verstanden wird. Auch wird die Eingabe sofort in eine Variable

übernommen, was Manipulationen mit einzelnen Zeichen überflüssig macht.
Ein gravierender Nachteil jedoch besteht darin, daß bei Verwendung einer numerischen Variablen als Input-Argument die Eingabe eines nicht numerischen Zeichens zum Programmabbruch führt. Das ist besonders unangenehm, wenn Benutzer und Programmierer nicht identisch sind, da der Anwender nicht wissen kann, wo das Programm zum ordnungsgemäßen Fortgang wieder aufgesetzt werden muß. Hier besitzt der
GET
einen klaren Vorteil, denn dieser übernimmt die Tastendrücke kritiklos, und das ist auch schon alles, was an Vorzügen zu berichten ist.
Nachteilig ist, daß beim Get kein Cursor erscheint. Die Aufforderung zur Eingabe muß also anderweitig erfolgen, entweder durch vorhergehenden Print einer Zeile oder durch Einschalten des Cursor. Außerdem muß ein String aus den einzelnen Zeichen zusammengepuzzlet werden, und es besteht keine Korrekturmöglichkeit während der Eingabe. Wir haben eine kleine Routine vorbereitet, die in Verbindug mit dem Maskengenerator diese Probleme zu Ihrer Zufriedenheit löst.

Die Frage, wann die eingegebenen Daten am Bildschirm angezeigt werden sollen, stellt sich eigentlich nur beim Get. Dieser bringt die Eingabe nicht automatisch, wie der Input, zur Anzeige. Es wird eigentlich blind eingegeben. Das macht es notwendig, das empfangene Zeichen sofort auf den Bildschirm zu printen, da eine Anzeige erst des fertig zusammengebauten Strings von großem Vertrauen in die Treffsicherheit des Benutzers zeugt.

Die Plausibilitätskontrolle schließlich ist ein Thema, dem viel zu wenig Beachtung geschenkt wird. Es läßt sich nämlich oft schon gleich nach der Eingabe überschlägig feststellen, ob ie Daten innerhalb von Ihnen festzulegender Wahrscheinlichkeitsgrenzen liegen, sei es durch Kontrolle der Länge eines Strings oder des Zahlenbereiches einer numerischen Eingabe. So ist es z.B. unwahrscheinlich, daß

eine Hausnummer 5 Stellen umfaßt oder das Geburtsdatum einer lebenden Person vor 1890 liegt. Sie könnten sogar während der Eingabe schon prüfen, ob nur Ziffern eingegeben werden, wenn solche auch verlangt sind. Mit Get wäre das, ungeachtet der sonstigen Nachteile prinzipiell möglich, etwa in der Form

```
10get a$:if a$=""then10
20if asc(a$)<48 or asc(a$)>57then10
```

Mit dieser Zeile lassen Sie nur die Ziffern 0..9 zu. Sie könnten den Bereich noch auf den Punkt und das Minuszeichen ausdehnen, die ja eigentlich auch zu den numerischen Eingaben zu zählen sind, oder auch nur bestimmte Buchstaben zulassen. Ihrer Phantasie sind keine Grenzen gesetzt.
Da aber kaum jemand sich die Mühe macht, haben wir diese Möglichkeiten in einer Input-Routine zusammengefaßt, die sowohl die Zeichenart (numerisch oder nicht) als auch die Menge der eingegebenen Zeichen überwacht.
In Verbindung mit dem Maskengenerator können Sie zuverlässig verhindern, daß im Feld Postleitzahl ein Buchstabe eingegeben wird, ja der Anwender wird sogar gezwungen, 4 Ziffern einzugeben. Vorher wird die Routine nicht verlassen. Was das für Sie heißt, können Sie selbst ermessen. Im Abschnitt 2.1.2 haben wir Ihnen gezeigt, welche Möglichkeiten Ihnen zur Definition eines Eingabefeldes zur Verfügung stehen. Die folgende Routine sorgt nun dafür, daß Ihre Eingabevorschriften streng beachtet werden. Sie werden zugeben, daß ein solches Verfahren etliche Fehlermöglichkeiten erst gar nicht aufkommen läßt.
Die Handhabung besteht einfach darin, daß Sie nach dem Maskenaufbau beliebig oft und für jedes Feld diese Routine mit

```
100f%=2:gosub49000
```

anspringen können. Sie brauchen nur f% mit der Nummer des gewünschten Eingabefeldes versorgen, und das Ergebnis wird Ihnen in f$ zurückgeliefert.

Beachten Sie bitte: Bei numerischen Feldern müssen sie f$ zuerst mit z.B. a=val(f$) umwandeln, wenn Sie mit dieser Zahl rechnen wollen.

```
49000 rem
49020 rem***************** feldeingabe ****************
49040 rem
49060 f$="":iff%(f%,0)=0thenreturn
49080 iff%<1orf%>23thenreturn
49100 printchr$(5);
49120 ml=f%(f%,1)and1
49140 cb$="                                        "
49160 mn=1:iff%(f%,1)and2thenmn=0
49180 il=f%(f%,0)
49200 cz=f%:cs=f%(f%,2)
49220 poke214,cz:poke211,cs:sys58732
49240 f$="":cc=0:ms=0
49260 poke204,0:rem cursor ein
49280 getg$:ifg$=""then49280
49285 ifasc(g$)<>13then49300
49290 iff%(f%,3)=1thencc=il
49295 goto49380
49300 printleft$(cb$,il);
49320 poke211,cs:f%(f%,3)=0
49340 goto49380
49360 getg$:ifg$=""then49360
49380 g=asc(g$):ifg=13thenonml+1goto49640,49600
49390 ifg>132andg<141thenf$=g$:cc=0:goto49640
49400 ifg=29andpeek(211)<=cs+il-1thengosub49820:goto49360
49420 ifg=157andpeek(211)>csthengosub49820:goto49360
49440 onmngoto49520
49460 if(g>47andg<58)org=45org=46then49540
49500 goto49360
49520 ifg<32or(g>127andg<160)then49360
49540 ifcc<ilandpeek(211)<=cs+il-1thencc=cc+1:
      gosub49820:goto49360
49560 ifpeek(211)<cs+ilthengosub49820
49580 goto49360
```

```
49600 iff%(f%,3)then49640
49620 ifpeek(211)<>f%(f%,2)+ilorcc<ilthen49360
49640 poke205,2
49660 ifpeek(207)<>0then49660
49680 poke204,1:rem cursor aus
49700 poke211,cs:poke214,cz
49720 ifcc=0thenreturn
49730 open3,3
49740 get#3,g$:ifg$=chr$(13)thenf$=left$(f$+cb$,il):return
49760 f$=f$+g$
49780 iflen(f$)<ilthen49740
49800 close 3 : return
49820 poke205,2
49840 ifpeek(207)<>0then49840
49860 printg$;:ifpeek(211)>msthenms=peek(211)
49880 return
```

Folgende Variable sollten Sie kennen:

f% muß die Feldnummer enthalten.
f%(f%,x) enthält die das Feld beschreibenden Parameter
cc enthält die Anzahl der gültigen Zeichen
il entspricht der Feldlänge
f$ schließlich enthält die fertige Eingabe

49000-
49260 Die Laufvariablen werden dem Array f%(f%,x) entnommen, der Cursor auf die erste Stelle des Eingabefeldes gesetzt und eingeschaltet.

49280-
49360 Ein Zeichen wird von der Tastatur geholt. Falls es das erste Zeichen war, wird eine evtl. Vorbesetzung des Feldes gelöscht.

49380-
49580 Ist das Zeichen RETURN (13), wird zur Überprüfung der Länge gesprungen. Die Zeichen Cursor rechts/links werden unmittelbar ausgeführt, falls das Zielfeld innerhalb des Eingabebereiches liegt (Kontrolle durch

peek(211)).
Falls es sich um Funktionstasten handelte, wird ohne weitere Überprüfung die Routine verlassen. Alle anderen Zeichen durchlaufen ein Filter (numerisches Filter in 49460, Steuerzeichensperre in 49520). Abschließend wird wieder geprüft, ob sich der Cursor noch innerhalb der Grenzen befindet.

49600-
49620 Hier wird die Zwangslänge geprüft. Wurden nicht genügend Zeichen eingegeben oder war das Feld nicht vorbesetzt, geht es wieder zurück zur Eingabe.

49640-
49800 Wurde kein Zeichen eingegeben, wird der Cursor ausgeschaltet und zurückgesprungen. Ansonsten werden die eingegebenen Zeichen in der Länge von il vom Bildschirm geholt und in f$ dem Benutzer übergeben.

49820-
49880 Diese kleine Subroutine dient der Anzeige des eingegebenen Zeichens, falls es die Filter passiert hat. Dazu wird die Dunkelphase des Cursor abgewartet (207=0), da sonst u.U. der Cursor am alten Platz stehenbleibt. Das Timing dafür ist in Basic nicht so genau hinzubekommen. Es kann daher schon einmal vorkommen, daß ein reverses Feld auf dem Schirm stehenbleibt. Der Gültigkeit der Daten tut das jedoch keinen Abbruch.

2.2 DATENVERARBEITUNG

Verehrter Leser, wir sind uns völlig klar darüber, daß wir mit diesem Abschnitt ein heißes Eisen anfassen und wir sind der faulen Eier, die Sie nach uns werfen werden, bereits gewärtig, aber sei's drum.
Bitte verstehen Sie uns nicht falsch:
Wir wollen hier nicht schulmeistern, denn wir sehen uns nicht als das Maß der Dinge, aber vielleicht zeigt Ihnen die eine oder andere Ausführung neue Wege auf, an die Sie bisher garnicht dachten, und damit haben wir unser Ziel eigentlich

schon erreicht.
Sicher hat jeder seinen Stil, und den möchten wir Ihnen auch nicht nehmen. Es könnte jedoch der Fall eintreten, daß Die ein besonders großes Programm erstellt haben, dem hinterher zum Ablauf der Soeicherplatz fehlt, weil Sie mit den Variablen allzu großzügig umgegangen sind, und damit sind wir auch gleich beim ersten Punkt unserer Betrachtungen.

2.2.1 VARIABLE

Variable, insbesondere numerische Variable, sind die Spielwiese aller Programmierer. Wie auch anders, da Sie ja nicht Programme in der im Abschnitt 2.1 beschriebenen Form produzieren wollen.
Haben Sie sich eigentlich schon einmal Gedanken darüber gemacht, wieviel Platz eine Variable im Speicher benötigt?
Gleitpunktvariable (das sind die ohne Zusatz) belegen fünf Bytes, den Platz für den Namen (2 Bytes) nicht mitgerechnet. denn den brauchen alle Variablentypen.
Damit überstreichen sie einen Bereich mit neun signifikanten Stellen (dei Stellen ohne Vor- und Nachnullen) und einem zweistelligen Exponenten.
Variable vom Typ Integer (%) haben einen Zahlenraum von +/- 32767, belegen aber den gleichen Platz wie Gleitpunktvariable, d.h. numerische Variable belegen grundsätzlich 7 Bytes.
Stringvariable ($) belegen zunächst auch 7 Bytes, nämlich zwei für den Namen, eins für die Länge und zwei Bytes als Pointer, welcher auf den Speicherplatz zeigt, wo die Nettodaten stehen. Dort wird für jedes Zeichen des Strings ein Byte belegt.
Wären noch die Arrays zu nennen. Das sind Variablenfelder, die mit einer DIM-Anweisung eingerichtet werden müssen.
Hier sieht der Platzbedarf etwas anders aus:
Gleitpunktarrays belegen nach wie vor pro Element fünf Bytes, jedoch Integerarrays nur zwei Bytes für jedes Element.
Stringarrays belegen zunächst 3 Bytes für jedes Element,

nämlich wieder eines für die Länge und zwei als Pointer. Die Nettodaten selbst werden wieder dynamisch angelegt.
Beachten Sie eines: Numerische Arrays werden sofort bei DIM in der vollen Größe angelegt, Stringarrays jedoch nur mit den Verwaltungseinträgen. Sie wachsen danach erst mit dem Füllen.

Was wir damit sagen wollen?
Bei numerischen Variablen ist es ziemlich gleichgültig, ob sie vom Typ Gleitpunkt oder Integer sind, jedoch bei Arrays sollten Sie sich überlegen, ob Sie nicht auch mit ganzen Zahlen bis 32767 auskommen können.
Eine äußerst platzsparende Methode für die Ausgabe auf Diskette (jedoch etwas rechenaufwendiger) ist die, ganze Zahlen in einer Stringvariablen unterzubringen. Bei Zahlen bis 255 geht das einfach durch die Anweisung

```
a$=chr$(a)
```

Die Rückverwandlung geschieht durch

```
a=asc(a$)
```

Von dieser Möglichkeit haben wir bei der Verwaltung der QUISAM-Datei reichlichen Gebrauch gemacht, um Platz auf der Floppy zu sparen. Zur Bearbeitung im Speicher eignet sich dieses Verfahren nicht, da hier mehr Platz gebraucht wird als bei einer numerischen Variablen.
Auch bei Arrays läßt sich gewaltig Platz sparen.
Angenommen, Sie benötigen ein Feld der Form a(1000), sind aber sicher, daß davon nur wenige Felder tatsächlich belegt werden, die Dimension aber trotzdem sein muß, da Sie nach einem bestimmten Algorhitmus auf die Felder zugreifen.
Probieren Sie in einem solchen Falle einmal ein Feld der Form a$(1000). Die Einträge nehmen Sie mit a$(x)=str$(a) vor. So belegen Sie nur den tatsächlich benötigten Platz.
Hervorholen können Sie die Daten wieder mit a=val(a$(x)).

Haben Sie überhaupt schon einmal daran gedacht, auf Floppy zu speichernde Daten zu komprimieren?
Mit Zahlen beliebiger Länge, sofern sie keine Nachkommastellen besitzen, läßt sich das hervorragend machen. Über je zwei Stellen der Zahl wird einfach der chr$

[illegible] sparen Sie die Hälfte des Platzes, was bei [illegible] Auftreten von Zahlen sehr viel sein kann.

Noch ein Tip:

Häufig wird, um etwa vorhandene Nachkommastellen abzuschneiden, die Funktion INT bemüht. Falls Sie sicher sind, daß die zu behandelnde Zahl den Bereich bis 32767 nicht verläßt, nehmen Sie doch anstelle von

a=int(b)   einfach einmal

a%=b

Das geht zum einen etwas schneller und spart auch Platz im Programmspeicher, weil die Anweisung kürzer ist.

Vielleicht überlegen Sie bei Ihrem nächsten Projekt einmal, was von dem Gesagten für Sie anwendbar ist.

2.2.2 RECHENGENAUIGKEIT

Es gibt beliebte Beispiele, die von Vertretern bestimmter Rechnerfabrikate immer wieder angeführt werden, um das Konkurrenzprodukt madig zu machen.

Eines davon ist eben

```
printsqr(5)^2
```

Was glauben Sie, was herauskommt?

5.00000001

Wie Sie sehen, hat der Rechner in der letzten Stelle einen Fehler gemacht.

Es gibt noch ein Beispiel:

```
10a=1
20a=a-.1
30printa
40goto20
```

Man sollte glauben, daß nun die Zahlen .9, .8, .7 usw. auf dem Bildschirm Revue passieren. Weit gefehlt. Bis .2 geht alles gut, dann kommt es:

.0999999998

-2.47382559E-10

Diese Effekte rühren daher, daß eben das Betriebssystem nur mit einer endlichen Anzahl von Stellen rechnet. Da nur neun

Stellen zur Anzeige gelangen, kann ein Fehler in der letzten Stelle schon empfindlich stören, insbesondere bei Kettenrechnungen, wo sich derartige Fehler addieren können, wonach das Endergebnis dann grob falsch sein kann.
Um Abhilfe zu schaffen, gibt es leider kein Patentrezept.
Das gegebene Verfahren wäre Runden, aber die Frage ist, ob auf oder ab. Der Fehler kann in beiden Richtungen auftreten.
Wie Sie vorgehen, wenn die letzte Stelle für Sie von Interesse ist, können nur Sie entscheiden, weil das abhängig von der vorzugsweise verwendeten Rechenart und natürlich Ihrer Anwendung ist, ob Sie lieber auf- oder abrunden.
Die Frage ist auch, an welcher Stelle gerundet werden soll, und das ist von Fall zu Fall verschieden. Die kaufmännische Praxis erlaubt das Runden in der dritten Stelle hinter dem Komma, während für wissenschaftliche Anwendungen (wenn z.B. mit der Wellenlänge des Lichts gerechnet werden soll) auch die neunte Stelle nicht akzeptabel ist.
Sie sehen, wir geben den Schwarzen Peter an Sie zurück.
Diesmal haben wir kein Progrämmchen als Problemlöser parat, jedoch wollten wir es nicht versäumt haben, Ihre Gedanken in die richtigen Bahnen zu lenken, wenn Sie einmal mit den Rechenergebnissen Ihres Programmes nicht so ganz einverstanden sind.

2.2.3 SORTIEREN

über dieses Thema ließe sich in ganzes Buch schreiben, denn es gibt hierfür fast soviele Verfahren, besondere Varianten davon und spezielle Winkelzüge, wie es Softwareautoren gibt.
Das Kernproblem besteht im Grunde nur darin, Daten (allgemein) nach einem bestimmten Kriterium zu sortieren, sei es auf- oder absteigend, oder in der ersten Hälfte das eine, in der anderen Hälfte das andere.
Bestimmt wird Ihnen auf Anhieb auch eine Lösung einfallen.
Was jeoch die Gemüter erhitzt, ist einfach die Frage, wie das am schnellsten zu bewerkstelligen ist.
Unser Bestreben ist es jedoch nicht, der Palette eine weitere Spielart hinzuzufügen (das überlassen wir

anschließend Ihnen), sondern Ihnen zwei gebräuchliche Sortieralgorhitmen vorzustellen, die Sie durchaus noch Ihrem persönlichen Geschmack anpassen können. Diese vorgestellten Verfahren bilden in aller Regel auch die Grundlage der Varianten, sodaß es Ihnen genügen mag, den Rumpf kennezulernen.

Das erste Verfahren, von dem hier die Rede sein soll, macht eigentlich nichts anderes, als was gewöhnlich auch Spieler mit ihren Karten tun: sie nehmen hohe Werte von rechts und stecken sie nach links auf die Hand und umgedreht. Das geschieht völlig unbewußt, ohne daß man sich dabei Gedanken über den Algorhitmus macht. Man weiß eben, wie es geht.

Das Ganze wird verbreitet auch Bubble-Sort genannt, angelehnt an Luftblasen, die in einer Flüssigkeit nach oben steigen und dem schwereren Medium unten Platz machen. Erinnern Sie sich nocheinmal an das Kartenspiel: durch das Bewegen der hohen Karten nach links rutschen die niederen Werte fast von alleine nach rechts. Das wird sooft wiederholt, bis sich die richtige Reihenfolge eingestellt hat.

Unserem Rechner fehlt natürlich der überblick, den Sie über Ihre Karten haben. Sie sehen sofort, wo das As steckt und packen es zielstrebig. Ein Computer müßte, von einer Seite beginnend, jede Karte einzeln anschauen und sie mit jeder der folgenden vergleichen, um festzustellen, welchen Wert die gerade betrachtete Karte im Vergleich zu den anderen hat. Dazu braucht er bei sieben Karten überschlägig 49 Vergleiche (7^2), weil jede Karte mit jeder anderen zu vergleichen ist, um auch ja keine auszulassen.

Damit ist die Funktionsweise des Bubblesort bereits so hinreichend umrissen, daß wir uns gleich an das zugehörige Programm wagen können. Sie werden erstaunt sein, wie kurz es ist.

Es besteht im wesentlichen aus zwei Zählschleifen, deren Maximalwert die Anzahl der Karten ist, nämlich einer inneren Schleife (L2) für den Vergleich der Karte L1 mit allen anderen Karten L2, und natürlich einem Stapel Karten (sd$(x)).

Nach einem L2-Durchlauf wird die nächste Karte L1 genommen

und wieder mit allen anderen Karten L2 verglichen.
Wie nun sortiert wird (vergleichen alleine hilft ja noch nicht)?
Ganz einfach: immer wenn festgestellt wird, daß eine L1-Karte größer als eine L2-Karte ist, werden die beiden kurzerhand vertauscht, indem die L1-Karte auf den Tisch (sa,sa$) gelegt wird, die L2-Karte an den alten Platz der L1-Karte kommt, diese wieder vom Tisch genommen und an den Platz der L2-Karte gesteckt wird.
Probieren Sie es ruhig einmal mit einem Kartenspiel aus. Sie weden sofort merken, worum es geht und worauf zu achten ist, um evtl. schneller zum Ziel zu kommen.
Das Sortierprogramm ist als Unterprogramm gestaltet. Zur Verfügung brauchen Sie eigentlich nur Ihre DAten zu stellen, und zwar in sd$(x). Dieses Feld müssen Sie selbst nach Bedarf dimensionieren. Damit der Sort feststellen kann, wie groß es ist, muß das letzte Element des Feldes chr$(255) enthalten.
Automatisch wird noch ein zweites Feld sd%(x) mitsortiert. Das hat folgende Bewandnis:
Aus längeren zu sortierenden Datensätzen, z.B. Adressen, ist in der Regel nur ein kleinerer Teil als Sortierkriterium interessant, beispielsweise der Nachname. Nur mit diesen füllen Sie sd$(x). In sd%(x) könnten Sie jetzt die zugehörige Satznummer der Diskette hinterlegen. Bubblesort macht den Vergleich zwar nur über sd$(x), jedoch wird sd%(x) auch vertauscht, sodaß Sie, falls es Ihnen nicht nur um den Inhalt von sd$(x) zu tun ist, sondern um den gesamten Satz, diese Sätze in der richtigen Reihenfolge hervorholen können, indem Sie Diskette einfach mit dem Inhalt von sd%(x) adressieren, beginnend bei sd%(0) bis zum Maximum. Dieses Maximum erhalten Sie aus der Variablen sn, welche beim Absuchen von sd$(x) nach chr$(255) als Zähler diente.
Nach dieser ausführlichen Einleitung nun das Programm, das wir seiner Kürze wegen nicht weiter kommentieren.
Ein wichtiger Hinweis: Sie müssen sd%(x) auch dann dimensionieren, wenn Sie keine korrespondierenden Einträge benötigen. Bubblesort rechnet auf jeden Fall damit. Andernfalls bekommen Sie einen 'bad supscript error'.

Falls Sie diese Funktion überhaupt nie benötigen, entfernen Sie die Zeilen 60240..60280.

```
60000 rem
60001 rem************* bubblesort *********
60002 rem
60060 sn=0:l1=255
60080 ifasc(sd$(sn))<>l1thensn=sn+1:goto60080
60100 sn=sn-1
60120 forl1=0tosn-1:forl2=l1+1tosn
60140 ifsd$(l1)<sd$(l2)then60300
60180 sa$=sd$(l1)
60200 sd$(l1)=sd$(l2)
60220 sd$(l2)=sa$
60240 sa=sd%(l1)
60260 sd%(l2)=sd%(l1)
60280 sd%(l1)=sa
60300 next:next
60320 return
```

übrigens: Wenn Sie eine absteigende Reihenfolge wünschen, brauchen Sie nur den Vergleich in Zeile 60140 umzukehren.
Als Größenordnung für den Zeitbedarf gilt etwa, daß zum Sortieren von 100 Elementen ca. 175 sec benötigt werden. Die Zeit steigt mit der Anzahl der zu sortierenden Elemente quadratisch an, d.h. für 1000 Elemente brauchen Sie bereits 17500 sec.
Wie es auch schneller geht, zeigt das nächste Verfahren, nämlich der

QUICKSORT
Dieser trägt seinen Namen zu Recht, denn er ist das durchschnittlich (zu 'durchschnittlich' noch eine Bemerkung am Ende) schnellste Verfahren. Um Ihnen einen Vorgeschmack zu geben: für 100 Elemente braucht Quicksort nur ca. 25 sec., also 7mal schneller als Bubblesort. Der Zeitbedarf wächst mit der Elementezahl nur etwas stärker als linear,

aber bei weitem nicht quadratisch. Dennoch ist das Programmchen etwa gleichgroß wie Bubblesort.

Wie das funktioniert?

Nehmen Sie wieder Ihr Kartenspiel von vorhin auf. Haben Sie eben jede Karte mit allen anderen verglichen, ob sie größer oder kleiner ist, so merken Sie sich nun eine Karte aus der Mitte der Anordnung. Jetzt vergleichen Sie zunächst die Karten links davon,vom linken Rand beginnend, mit der mittleren. Haben Sie eine gefunden, die kleiner ist, merken Sie sich diese und vergleichen nun die Karten der rechten Hälfte mit der mittleren, vom rechten Rand beginnend. Entdecken Sie eine, die größer ist als diese, tauschen Sie einfach die gefundene Karte mit der, die Sie sich vorhin gemerkt haben, aus.

Nun fahren Sie bei der linken Seite da fort, wo Sie vorhin unterbrochen haben. Werden Sie wieder fündig, merken Sie sich auch diese Karte und machen rechts weiter, und so fort, bis sich der recht und der linke Vergleich treffen (das muß durchaus nicht in der Mitte sein. Wenn Sie nun die Karten anschauen, können Sie schon sehen, daß sich links von dem gedachten Treffpunkt alle Karten befinden, die größer sind als die ursprünglich willkürlich zum Vergleich herangezogene, und rechts davon alle kleineren. Das ist schon ein gewaltiger Fortschritt, weil das Kartenspiel jetzt grob vorsortiert ist. Nun merken Sie sich wieder etwa aus der Mitte zwischen rechtem Rand und dem Treffpunkt eine Karte und führen das oben beschriebene Verfahren mit dem rechten Viertel und dem zweiten Viertel von rechts durch. Auf diese Weise behandeln Sie die immer kleiner werdenden Teilstücke, bis nur noch zwei Karten übrig sind. Nun hat die zuletzt behandelte, ganz rechts außen liegende Teilmenge die richtige Reihenfolge und Sie wenden das Verfahren auf die nächste weiter links liegende kleinste Teilmenge an, bis Sie wieder beim allerersten Treffpunkt gelandet sind. Alle Karten rechts davon haben nun die richtige Reihenfolge. Nun werden die Karten links davon sortiert, immer kleinere Teilmengen bis zum linken Rand bildend und dann von der letzten ausgehend wieder zur Mitte hin. Jetzt ist auch die linke Seite vollständig sortiert. Da aber im ersten

Durchlauf bereits alle Karten, gemessen an der gedachten Vergleichskarte, nach rechts und links verteilt wurden, muß nun zwangsläufig das gesamte Kartenspiel sich in der richtigen Reihenfolge befinden.
Das hört sich zwar ungeheuer kompliziert an, aber wenn Sie es einmal durchführen werden Sie merken, wie einfach das Verfahren im Grunde ist, da immer die gleiche Behandlung immer kleineren Mengen zuteil wird.
Auch programmtechnisch gibt es keine Schwierigkeit und es werden auch dieselben Eingangsparameter gebraucht. Um bei der bildlichen Vorstellung zu bleiben:

sd$ (nicht sd$(x)!) enthält die zum Vergleich aus der Mitte gemerkte Karte.

sv$, bzw. sv ist der Tisch, auf dem wir eine Karte beim Vertauschen ablegen (wir haben ja nur zwei Hände).

s1(x) enthält alle linken Ränder der noch zu bearbeitenden Teilmengen.

s2(x) enthält alle rechten Ränder der noch zu bearbeitenden Teilmengen.

s1 ist der linke Rand des gerade bearbeiteten Teils.

s2 ist dessen rechter Rand.

sp ist ein Zeiger auf die 'Ränderspeicher'. Beim Zurückarbeiten vom Rand nach innen wird hiermit nach Abhandlung der Teilmenge ein neues Ränderpärchen aus den Speichern geholt, wo sie natürlich in derselben Reihenfolge stehen, wie sie zwecks Abarbeiteung der nächstkleineren Menge zunächst dort hinterlegt wurden.

l1 und l2 schließlich enthalten die Zeiger auf die gerade zum Vergleich mit der mittleren herangezogenen Karten.

Damit halten wir das nun folgende kurze Programm auch schon für hinreichend erläutert. Hier ist es:

```
61000 rem
61001 rem***************** quicksort *****************
61002 rem
61020 onsf%goto61060
61040 dims1(15),s2(15):sf%=1
61060 sn=0:l1=255
61080 ifasc(sd$(sn))<>l1thensn=sn+1:goto61080
61100 sn=sn-1:sp=0
61120 s1(0)=0:s2(0)=sn
61140 s1=s1(sp):s2=s2(sp)
61160 sp=sp-1
61180 l1=s1:l2=s2
61200 sd$=sd$((s1+s2)/2)
61220 ifsd$(l1)<sd$andl1<s2thenl1=l1+1:goto61220
61240 ifsd$(l2)>sd$andl2>s1thenl2=l2-1:goto61240
61260 ifl1>l2then61340
61280 sv=sd%(l1):sd%(l1)=sd%(l2):sd%(l2)=sv
61300 sv$=sd$(l1):sd$(l1)=sd$(l2):sd$(l2)=sv$
61320 l1=l1+1:l2=l2-1
61340 ifl1<=l2then61220
61360 ifs2-l1<=l2-s1then61420
61380 ifs1<l2thensp=sp+1:s1(sp)=s1:s2(sp)=l2
61400 s1=l1:goto61460
61420 ifl1<s2thensp=sp+1:s1(sp)=l1:s2(sp)=s2
61440 s2=l2
61460 ifs2>s1then61180
61480 ifsp>-1then61140
61500 return
```

Wir haben Ihnen nun eines der langsamsten und eines der schnellsten Sortierverfahren vorgestellt. Diese Klassifizierung ist als durchschnittlich zu verstehen. Durchschnittlich heißt hier, daß die Programme sowohl auf zufällige oder bereits sortierte oder absichtlich falsch sortierte Elemente losgelassen werden, wobei sich in jedem Falle unterschiedliche Zeiten ergeben, von denen der Durchschnitt ermittelt wird.
Wenn Sie den Bubblesort derart optimieren, daß ein

Schleifendurchlauf ohne Vertauschung von Elementen einen Merker setzt, können Sie auch dieses Verfahren erheblich abkürzen. Dann hat es auch seine Berechtigung als einfacher Algorhitmus zum Sortieren weniger Elemente (bis etwa 10).
Bei großen Feldern ist der Quicksort Meister aller Klassen.

In diesem Zusammenhang vielleicht noch ein paar Betrachtungen zum Auffinden bestimmter Elemente aus einem Feld.
Ging es beim Sortieren darum, eine gewisse Ordnung herzustellen, ist bei der Suche ein möglichst schneller Zugriff erwünscht. Die Prinzipien der linearen Vorgehensweise wie beim Bubblesort oder die der Teilmengenbildung wie beim Quicksort sind auch hier anzuwenden. Welches Verfahren bei der Suche das schnellere (oder überhaupt mögliche) ist, hängt einzig und allein von der Anordnung der Elemente ab. Liegen diese sortiert vor, bietet die Teilmengenbildung erhebliche Vorteile, da wieder zunächst der Suchbegriff mit dem mittleren Eintrag verglichen wird. Ist der Suchbegriff kleiner, wird aus der Mitte der unteren Hälfte ein Vergleichskriterium hergenommen, usw., bis die Auswahl nur noch zwischen zwei Elementen erfolgt. Dieses Verfahren ist keinesfalls auf unsortierte Elemente anwendbar, wie sich schon aus dem Verfahren selbst ergibt.
Die lineare Suche, nämlich von unten nach oben (oder umgekehrt), ist immer möglich, dauert jedoch naturgemäß am längsten, wenn der zu suchende Eintrag ausgerechnet am anderen Ende der Anordnung steht.
Ihnen hierfür ein Programm anzubieten, könnten Sie als Beleidigung auffassen. Deshalb nehmen wir davon lieber Abstand.

2.3 DATENAUSGABE

Eine Frage, die erfahrungsgemäß viel zu spät, nämlich erst, wenn das Programm fast fertig ist, gestellt wird, ist die nach dem WOHIN?.
Sie erfreuen sich gerade noch an den raffinierten Winkelzügen, mit denen Sie noch ein paar Sekunden Laufzeitersparnis herausgekitzelt haben, alle Algorhitmen arbeiten zur vollen Zufriedenheit, da trifft Sie der Blitz:
Sie haben während der ganzen Aktion die Datenausgabe ziemlich außer acht gelassen, weil Sie nämlich (was richtig ist) einzelne Module und Unterprogramme auf ihre Funktionstüchtigkeit getestet haben, diese dann zu einem Ganzen gebunden und dabei versäumt, evtl. zur Ausgabe gelangende Daten rechtzeitig und in geeigneter Form bereitzustellen.
Zum Zeitpunkt des Auftretens wäre das noch bequem möglich gewesen, da dort evtl. noch weitere beschreibende Parameter vorhanden waren. Hierbei brauchen Sie nur einmal an die Definition eines Eingabefeldes im Abschnitt 2.1.2 zu denken. Bei der Eingabe von Daten wußten Sie noch, wie lang der String war, weil das zugehörige Feld entsprechend dimensioniert war. Nun aber, da Ihnen die Daten nur als nackter String vorliegen, müssen Sie die Länge zwecks vielleicht formatgerechter Ausgabe erst ermitteln.
Was wir damit sagen wollen, ist einfach folgendes:
Sie sollten zu jedem Zeitpunkt des Auftretens von wie auch immer gearteten Daten prüfen, ob und in welcher Form sie für eine Ausgabe benötigt werden, damit Sie diese, um bei unserem Beispiel von oben zu bleiben, auf das geforderte Format auffüllen oder ausrichten können. Zu diesem Thema erfahren Sie noch mehr im Abschnitt 2.3.2.3.

Gegenstand der folgenden Betrachtungen werden zunächst die Ausgabe auf Floppy, und im Abschnitt 2.3.2 die Ausgabe auf Drucker sein.
Die Bildschirmausgabe, als Ergebnis eines Programmes gesehen, lassen wir außer acht, da der Bildschirm als naturgemäß flüchtiges Medium nur der vorübergehenden

Anzeige, wie z.B. bei der Maske, dient.

2.3.1 AUSGABE AUF FLOPPY

Wer sich zum ersten mal mit einer Floppy beschäftigt, wird wohl ein wenig fassungslos das zugehörige Handbuch studieren.
Von einer einfachen Syntax wie bei der Ausgabe auf Bildschirm oder (schon vielfältiger) Drucker kann hier nicht mehr die Rede sein. Beim OPEN geht es schon los. Das Einfachste und daher auch meistgebrauchte ist das Laden und Speichern von Programmen. Wer sich bisher mit einer Datasette herumgequält hat, weiß das auch zu würdigen.
Der nächste Schritt, weil am druckerähnlichsten in der Handhabung (abgesehen vom OPEN) ist eine serielle (oder auch sequentielle) Datei. Ab hier wird das Bedienungshandbuch für die meisten uninteressant, denn wer mag sich schon (von einigen Spezialisten einmal abgesehen) mit den Block-Direktzugriffen und der zugehörigen Freispeicherverwaltung herumschlagen.
Dennoch bietet die Floppy mehr, wenngleich bedauerlicherweise im Handbuch nicht beschrieben. Da wäre die Erweiterung sequentieller Dateien oder die Handhabung von REL-Files (was ist das schon wieder).
Auch diese Dinge sind vergleichsweise einfach, also ohne 'Bitfieselei', zu handhaben. Wie, das werden wir Ihnen zeigen.

2.3.1.1 SEQ

Die sequentielle Datei ist erstaunlicherweise die häufigst benutzte, wenn auch für die meisten Anwendungen, bei denen die Daten später wieder gebraucht werden, die weitaus unpraktischste.
Das liegt sicher zum einen daran, daß echte Alternativen nicht angeboten werden; zum anderen ist die Anwendung einfach zu beherrschen.
Was heißt nun SEQuentiell?
Sequentiell könnte einfach mit 'der Reihe nach' übersetzt werden, was auch den Kern der Sache trifft. Es ist in der Tat so, daß eingehende Daten im Augenblick des Auftretens (PRINT#-Befehl) auf der Stelle ohne Federlesens auf die Diskette (den floppyinternen Puffer unterschlagen wir) geschrieben werden.
Wollen wir diese Daten zurückholen, so kommen sie genau so unbesehen wieder in den Rechner.
Alles prima, denken Sie, wo liegt die Problematik?
Stellen Sie sich vor, Sie haben einhundert Meter Schnur und wollen diese zwecks späterer Verwendung aufwickeln. Sie drehen sie also auf das Knäuel, so wie Ihnen der Faden durch die Finger läuft. Mittendrin kommt Ihnen der Gedanke, ein kurzes Stück des Seiles rot zu färben. Anschließend wickeln Sie den Rest auf.
Eine Woche später benötigen Sie ausgerechnet das rote Stück. Da es sich mitten im Knäuel befindet, kommen Sie nicht daran. Es bleibt Ihnen nichts anderes übrig, als die ganze Schnur bis zu der roten Stelle abzuwickeln. Unpraktisch, nicht wahr?
Genauso verhält es sich mit einer Datei vom Typ SEQ und genau hierin liegt die Tücke.
Der Schnurwickel ist brauchbar, wenn Sie das Seil als Ganzes wieder benötigen. Sie merken sicher, worauf wir hinauswollen.
Wozu ist denn eine solche Datei überhaupt gut?
Sie dient zur schnellen Speicherung großer Mengen von Daten, die Sie später wieder <u>sämtlich</u> und in genau der <u>Reihenfolge</u> brauchen.

Eine Möglichkeit ist die in 2.2.5 beschriebene Rettung von Variablen.

Eine auch in professionellen Programmen geübte Praxis ist die Fortschreibung von Werten. Denken Sie sich einmal die Anwendung als Kasse. Alle Einnahmen werden der Reihe nach weggeschrieben. Abends brauchen die Inhalte der Datei nur nacheinander eingelesen und fortlaufend addiert werden, um den Tagesumsatz zu erhalten.

Um einmal bei der Fortschreibung zu bleiben:

Es kann rein theoretisch der Fall auftreten, daß zwischen den jeweils zu speichernden Daten größere Zeitintervalle auftreten. Sicherheitshalber werden Sie also die Datei nach jedem Zugriff wieder schließen. Wie aber neue Daten an die bestehenden anhängen?

Im Handbuch sind nur zwei Zugriffsarten auf die serielle Datei beschrieben, nämlich:

open lfn,"name,s,w" für Schreiben, und

open lfn,"name,s,r" für Lesen.

Versuchen Sie, eine schon bestehende Datei zum Schreiben zu eröffnen, wird das rote Lämpchen an der Floppy lustig blinken. Der Fehlerkanal bringt es an den Tag:

file exists error

Ganz Schlaue gehen hin, lesen die alten Daten herunter, schaufeln diese in eine Datei anderen Namens und schieben die neuen Daten hinterher. Am nächsten Tag dasselbe Spiel.

Es geht auch einfacher. Probieren Sie einmal folgendes:

open lfn,"name,s,a"

a steht hier für APPEND, also anhängen.

In dieser Zugriffsart werden neue Daten an die bereits bestehenden angehängt.

Warum das nicht im Handbuch steht?

Das wissen selbst wir nicht.

Um diesen Abschnitt zusammenzufassen:

Sequentielle Dateien eignen sich zur schnellen Speicherung von Daten, die später <u>allesamt</u> wieder benötigt werden.

Sie eignen sich nicht zur Ablage von Werten, auf die nachher jeweils <u>gezielt</u> zugegriffen werden soll.

2.3.1.2 REL

Ein neuer Begriff taucht hier auf. REL bedeutet RELative Datei.
Um wieder auf unser (hier hinkendes, jedoch die Zugriffsart verdeutlichendes) Beispiel mit der Schnur zurückzukommen:
Sie haben eine Reihe numerierter Fächer, in die jeweils genau ein Meter unserer Schnur hineinpaßt. Da Sie des öfteren verschiedenfarbige Schnüre von 1m Länge brauchen, haben Sie sich einige davon präpariert und auf die Fächer verteilt.
Damit Sie sich bei Bedarf nicht erst durch Augenschein von der Farbe überzeugen müssen (die Fächer sind geschlossen, Sie müßten sie erst öffnen), haben Sie sich ein Verzeichnis angelegt, aus dem der Zusammenhang von Fachnummer und Schnurfarbe hervorgeht. Sie brauchen also nur in Ihre Liste zu schauen, um die gewünschte Kordel mit <u>einem</u> Griff zu bekommen.
Hieraus wird zweierlei deutlich:

1. Es kommt nicht sosehr auf den speziellen Fachinhalt an (es könnten auch Schrauben und Radiergummis sein) als vielmehr auf die zu der Anordnung passende Liste. Denken Sie nur an den Bibliothekar, der ohne hinzuschauen das richtige Buch greift.
2. Jedes Fach einer Anordnung hat dieselbe Größe. Es passen entweder drei Radiergummies oder hundert Schrauben hinein.

Zurück zur Datenverarbeitung: Mit Radiergummies und Schrauben könnten beispielsweise Strings unterschiedlicher Länge gemeint sein.
Was bedeutet aber nun der Begriff 'relativ' in diesem Zusammenhang konkret?
Relativ heißt einfach 'bezogen auf den Dateianfang'.
Ein REL-File besteht aus einer Anzahl reservierter Speicherbereiche gleicher Länge. Damit Sie als Anwender wahlfrei auf einen dieser Bereiche zugreifen können, sei es zum Schreiben oder zum Lesen, ohne daß Sie wissen müssen, wo sich dieser Platz physikalisch auf der Diskette befindet,

verwaltet die Floppy intern sogenannte Datensatzzeiger. Wenn Sie also der Floppy mitteilen, daß Sie auf den Platz mit der Nummer 10 zuzugreifen wünschen, wird dies als der zehnte Platz, gezählt vom Dateianfang, also relativ zum Dateianfang, interpretiert. Der Zeiger wird auf diesen Satz gerichtet und, abhängig davon, ob das nachfolgende Kommando ein PRINT# oder ein INPUT# ist, der Bereich beschrieben oder gelesen.
Hieraus erkennen Sie, daß eine REL-Datei immer für beide Zugriffsarten geöffnet ist. Sie brauchen beim OPEN nicht zu spezifizieren, ob Sie schreiben oder lesen wollen.
Praktisch, nicht wahr?

Nun geht es los:
Wie teilen Sie der Floppy mit, wieviele Sätze welcher Länge Sie gerne reserviert hätten. Dem Handbuch ist, wie gesagt, nichts darüber zu entlocken. Das Problem liegt auch in der Tat nicht bei der Floppy, sondern im Betriebssystem des C64, der nicht über die komfortablen entsprechenden Kommandos des Basic 4 der 'großen' Commodore-Rechner verfügt.
Wir müssen diese Befehle also 'zu Fuß' generieren.
Dazu eine kleine Routine, die wir daran anschließend kommentieren werden:

```
63800 rem
63801 rem************* rel einrichten *********************
63802 rem
63810 ifrl%<2 or rl%>254 or r>65535 thenf1$="99":return
63820 close6:open6,8,6,"0:"+dn$+",l,"+chr$(rl%)
63840 rh%=r/256:rl%=r-256*rh%
63860 print#15,"p"chr$(6)chr$(rl%)chr$(rh%)chr$(1)
63880 print#6,chr$(255)
63900 input#15,f1$,f2$,f3$
63920 close6:rz$="":return
```

Es geht also, wie gesagt, darum, auf der Diskette Speicherplatz zu reservieren, den Sie in mundgerechten

Häppchen verarbeiten können.
Die Anzahl der Happen ist zunächst nicht wichtig, da sie erst bei Bedarf zur Verfügung gestellt werden brauchen. Vielleicht sollten wir diesen Punkt an dieser Stelle noch näher beleuchten: Wollen Sie einen Satz beschreiben, der an der Obergrenze des beabsichtigten Bereiches liegt, z.B. die Nummer 1000, so wird die Datei aufgebläht, d.h. alle bis dahin noch nicht belegten Datensätze bis 999 werden jetzt eingerichtet. Die Datei ist wesentlich größer geworden, als bis jetzt sinnvoller Inhalt darin steht. Das hat allerdings für die folgende Verarbeitung den Vorteil, daß die Bearbeitung schneller geht, da neue Sätze nicht jeweils eingerichtet werden müssen, sondern bereits, wenn auch leer, zur Verfügung stehen.
Was allerdings festgelegt werden muß, ist die Länge eines Satzes. Diese Information braucht die Floppy, damit beim 'Aufblähen' die richtige Speichergröße reserviert wird (überschlägig Satzlänge * höchste Satznummer).

Unsere Routine braucht natürlich Übergabeparameter, damit sie weiß, was womit zu tun ist. Diese seien zunächst vorgestellt.

dn$ Name, unter der die Datei eingerichtet werden soll.

rl% Länge eines Satzes. Dieser Wert muß wenigstens 2 und darf höchstens 254 betragen. Ansonsten reagiert die Floppy fehlerhaft. Weitere Verwendung von rl% siehe rh%.

r Hier wird die höchste Satznummer übergeben, auf die die Datei zunächst aufgeblasen werden soll. Die kleinste Satznummer ist 1, die größte 65535.
Ergeben sich im Laufe der Verarbeitung höhere Satznummern als zunächst eingerichtet, wird die Datei automatisch vergrößert, sofern überhaupt noch Platz auf der Diskette vorhanden ist. Limit für die Satznummer ist aber in jedem Falle 65535.

rh% Ohne näher auf die Sedezimalrechnung einzugehen sei hier nur soviel gesagt, daß der Wert r in zwei Bytes zerlegt wird, wovon rh% den höherwertigen Teil und rl% den niederwertigen Teil der Zahl r enthält.

f1$ Fehlerkennung (00 = alles klar)

f2$ Fehler im Klartext (z.B. file not found)

f3$ Erweiterung der Fehlermeldung (beim Löschen von Dateien steht hier z.B. die Anzahl der gelöschten Files)

Nun der Kommentar:

63810 Hier werden die Parameter rl% und r auf Gültigkeit überprüft. Sollte ein Wert ungültig sein, wird in f1$ ein Fehler simuliert. Sie sollten zweckmäßigerweise nach jedem Ansprung der Routine, das gilt auch für alle folgenden, stets auf val(f1$)=0 abfragen, um sich vom fehlerfreien Ausgang der Aktion zu überzeugen.

63820 Über die Sekundäradresse 6, den 'Datenkanal', übergeben wir anschließend die Drivenummer (hier 0), den Dateinamen, das Literal 'l' als Kennzeichen, daß eine REL-Datei eingerichtet werden soll, und anschließend die gewünschte Satzlänge.

63840 r wird in zwei Bytes zerlegt. Sie sehen hier eine Nutzanwendung aus 2.2.1. Dieser Trick erspart die Funktion INT.

63860 Hier 'blasen' wir die Datei auf, indem wir über den Kommandokanal das Kennzeichen 'p' für 'Satzzeiger positionieren' übergeben, dann die Information, über welchen Kanal die Daten laufen, dann unseren Satzzeiger (r) und zum Schluß eine Besonderheit, nämlich die relative Position innerhalb eines Satzes. Auch das ist möglich. Wir setzen diesen Bytezeiger jedoch immer auf 1, d.h. auf den Anfang des jeweiligen Satzes.

63880 Damit auch wirklich die Datei bis zur angegebenen Satznummer angelegt wird, muß wenigstens ein Zeichen übergeben werden. Wir wählen dazu das Zeichen pi (chr$(255), welches uns bei späteren Lesezugriffen einen leeren Satz kennzeichnen soll. Das macht übrigens die Floppy auch automatisch mit allen dazwischenliegenden Leersätzen.

63900 Durch Einlesen des Fehlerkanals erhalten wir Auskunft,

ob alles glatt verlaufen ist.

63920 Die Kanäle werden geschlossen. Hierdurch wird die floppyinterne Freispeicherverwaltung auf die aktuellen Werte gesetzt.

ACHTUNG: Irgendwann zu Beginn Ihres Programmes muß einmal der Befehl OPEN15,8,15 erfolgt sein, damit der Kommando- und Fehlerkanal gehandhabt werden kann. Ein CLOSE15 sollte erst bei Programmende erfolgen.

Eingerichtet wäre die Datei nun. Sie möchten sie sicher auch gerne mit Daten versehen. Im Grunde geht das nicht anders als bei einem Drucker, nur daß Sie vorher den Satzzeiger übermitteln, damit die nachfolgenden Daten auch auf den gewünschten Platz geschrieben werden.

Damit sich das für Sie auch recht komfortabel gestaltet, haben wir wieder ein kleines Unterprogramm vorbereitet, das wir Ihnen hier vorstellen wollen. Daran anschließend wieder die Variablen und der Kommentar.

```
63000 rem
63001 rem*************** rel schreiben ******************
63002 rem
63010 ifrz$=dn$then63040
63020 close6
63030 open6,8,6,"0:"+dn$:rz$=dn$
63040 rh%=r/256:rl%=r-256*rh%
63060 print#15,"p"chr$(6)chr$(rl%)chr$(rh%)chr$(1)
63080 print#6,r$
63100 input#15,f1$,f2$,f3$
63120 return
```

Zusätzlich zu den beim Einrichten beschriebenen Parametern tritt hier noch die Variable
r$ auf. Hierin werden die zu schreibenden Daten übergeben.

63010 Um Zeit zu sparen, wird der open übersprungen, falls

der letzte Zugriff auf dieselbe Datei erfolgt war.

63020 Auch hier wird wieder der Kommandokanal und der Datenkanal geöffnet. Allerdings braucht beim öffnen einer bestehenden Datei das Literal 'l' und die Satzlänge nicht mehr übergeben werden, da die Floppy diese Information aus dem Inhaltsverzeichnis entnehmen kann.

63040 Die Zerlegung von r

63060 Positionierung des Satzzeigers und des Bytezeigers

63080 Hier werden nun endlich die Daten geschrieben.

63100 Abfrage des Fehlerkanals

Jetzt haben Sie glücklich Ihre Daten auf ihren zugewiesenen Plätzen.

Fehlt nur noch das Wiederhervorholen. Auch hier müssen Sie vor dem Lesebefehl wieder den Satzzeiger laden, damit Sie auch die gewünschten Daten bekommen. Wie Sie sich denken können, haben wir auch hierfür ein Programmchen.

Hier ist es auch schon

```
63500 rem
63501 rem*************** rel lesen **********************
63502 rem
63510 ifrz$=dn$then63540
63520 close6
63530 open6,8,6,"0:"+dn$:rz$=dn$
63540 rh%=r/256:rl%=r-256*rh%:r$=""
63560 print#15,"p"chr$(6)chr$(rl%)chr$(rh%)chr$(1)
63580 input#15,f1$,f2$,f3$:iff1$<>"00"then63720
63590 ifrc%=255then63740
63600 forl1=1torc%
63620 get#6,ri$:ifri$=""thenri$=chr$(0)
63630 input#15,f1$,f2$,f3$:iff1$<>"00"thenl1=rc%:goto63680
63660 r$=r$+ri$
63680 nextl1
63700 ifleft$(r$,1)=chr$(255)thenf1$="50"
63720 return
```

```
63740 get#6,ri$:ifri$=""thenri$=chr$(0)
63760 input#15,f1$,f2$,f3$:iff1$<>"00"then63700
63770 ifri$=chr$(13)then63700
63780 r$=r$+ri$:goto63740
```

Die Variablen kennen Sie bereits aus den vorangegangenen Routinen bis auf:

ll Laufvariable

ri$ Interimsvariable zum Füllen von r$

rc% Anzahl der gewünschten Bytes aus dem Satz, wenn Sie weniger wollen, als der Satz lang ist. Wollen Sie den ganzen Satz, setzen Sie rc%=255, da die Routine sowieso nur bis CR (chr$(13)) liest, was das logische Ende des Satzes anzeigt.

Hier der Kommentar:

63520-
63580 Gleiche Funktion wie in der Schreibroutine, jedoch wird hier bereits der Fehlerkanal abgefragt, damit Sie merken, wenn Sie auf einen noch nicht angelegten Satz positioniert haben (f1$="50").

63600-
63680 Die Bytes werden einzeln hereingeholt und auf r$ addiert. Dieses Verfahren dauert zwar etwas länger als mit INPUT#, hat jedoch den Vorteil, daß auf den Datentyp keine Rücksicht genommen werden braucht (wie in 2.1.4) und außerdem auch Sonderzeichen eingelesen werden können, bei denen jeder INPUT abbrechen würde, nämlich z.B. das Komma und chr$(0).
r$ wird solange gefüllt, bis entweder das durch rc% vorgegebene Limit erreicht ist.
Selbstverständlich wird nach jedem Byte der Fehlerkanal abgefragt, um Lesefehler feststellen zu können.

63700 Wie oben angemerkt, enthält ein zwar eingerichteter, jedoch noch nicht beschriebener Satz als erstes

Zeichen chr$(255). Damit Sie einem leeren Satz nicht auf den Leim gehen, wird in f1$ der Fehler 50 (block not available) simuliert.

63740-
63780 Falls rc%=255 ist, wird nur solange gelesen, bis das Endekriterium chr$(13) erreicht ist.

Mit den vorgestellten drei Routinen können Sie nun schon fleißig auf REL-Files 'herumgeigen'. So kompliziert ist es doch garnicht, oder?
Wie bei allem, so gilt auch hier: Probieren geht über Studieren.

Abschließend noch einmal eine Zusammenfassung der Grundgedanken zu SEQ und REL anhand der Gegenüberstellung der gedachten Anwendungen Adressverwaltung und Datenkasse:
Bei einer Adressdatei geht es neben der Speicherung wesentlich um das Wiederauffinden der zu einem bestimmten Namen gehörenden übrigen Daten (wie Anschrift usw.). Wie Sie nun selbst beurteilen können, wäre es heller Wahnsinn, diese Daten in einem SEQ-File abzulegen, da das Hervorholen endlos dauern würde, da auch alle nicht gesuchten Einträge gelesen werden müssen. Sinnvoll ist hier einzig und allein eine Datei vom Typ REL, da hier jeder beliebige Eintrag anhand eines Verzeichnisses, welches Sie sich angefertigt haben und das nur Satznummer und Nachnamen enthalten braucht, gezielt und auf der Stelle eingelesen werden kann.
Auf der anderen Seite wäre es unsinnig, Fortschreibedaten in eine REL-Datei packen zu wollen, da der Platzbedarf bei gleichem Datenaufkommen höher als bei SEQ ist (die floppyinterne Verwaltung braucht bei REL-Files auch Platz auf der Diskette) und evtl. der Satzzeiger vor jedem Zugriff verändert werden muß, was Zeit kostet.
Eine Dateistruktur, die beider Vorteile glücklich vereint, sich mit vernünftigem Aufwand jedoch auf einer VC1541 nicht realisieren läßt, stellen wir Ihnen im nächsten Abschnitt vor.

2.3.1.3 ISAM

Um es gleich zu sagen: Diese Dateiart steht Ihnen auf Ihrer Floppy nicht zur Verfügung, jedenfalls nicht mit einer ähnlich bequemen Handhabung wie eine REL-Datei.
Warum wir uns dennoch darüber auslassen?
Wir sind der Meinung, daß die Kenntnis höherer Dateiformen nicht schaden kann. Sie soll Sie darüberhinaus zu eigenen Versuchen anregen, die in äußerst komfortablen Dateianwendungen ihren Niederschlag finden können.
ISAM steht für Indexed Sequential Access Method, was soviel bedeutet wie indizierte serielle Zugriffsmethode.
Darunter können Sie sich sicher auch noch nicht viel mehr vorstellen. Die Komponente 'seriell' fällt allerdings sofort ins Auge. Hatten wir doch schon. Richtig, das bedeutet doch eins nach dem anderen oder so ähnlich.
Was also soll an dieser Dateiform so exotisch sein?
Sie werden es nicht glauben, aber der Bestandteil 'indiziert' hat es in sich. Wieso, werden Sie gleich sehen.
Führen wir uns doch nocheinmal den Aufbau einer REL-Datei vor Augen, wie sie das Betriebssystem der Floppy VC-1541, aber nicht des C64 vorsieht (vergleichen Sie hierzu die entsprechenden "Kunstgriffe" zur Nutzung der relativen Dateien in unserem Floppy-Buch):
Ein REL-File besteht aus einer Anordnung von Datensätzen gleicher Länge, auf die mittels eines Schlüssels (haben wir gerade Schlüssel gesagt? Merken Sie sich den Ausdruck in diesem Zusammenhang), der in diesem Falle aus einer Nummer besteht, zugegriffen werden kann. Ein Zusammenhang zwischen Satznummer und Satzinhalt ist nicht gegeben, d.h. aus der Satznummer als solcher können keinerlei Rückschlüsse auf den Satzinhalt abgeleitet werden. Der Zusammenhang wird erst durch ein Inhaltsverzeichnis oder dergleichen hergestellt.
Beim fortlaufenden Lesen, d.h. ohne jeweils den Satzzeiger explizit zu versorgen, wird immer der Satz mit der nächsthöheren Nummer zur Verfügung gestellt ohne Rücksicht auf den logischen Zusammenhang der Satzinhalte.
So gesehen verhält sich eine relative Datei auch seriell und genauso ist auch der Terminus 'seriell' in einer ISAM-Datei

zu verstehen.

Wenn wir ISAM einen eigenen Abschnitt widmen, muß es sich schon gründlich von den anderen Dateiformen unterscheiden und das tut es auch, wenn es zunächst auch noch nicht so aussehen mag.
Wir haben oben den Ausdruck 'Schlüssel' so sehr betont, und genau das ist auch Aha-Erlebnis von ISAM.
Wenn wir im Zusammenhang mit REL-Dateien von einem Schlüssel sprachen, so ist das vergleichsweise so, als ob die Art eines Haustürschlüssels aus der Hausnummer ersichtlich wäre. Jeder, der die Hausnummer kennt, gelangt auch ins Haus hinein. Von einem echten Schlüssel kann dabei natürlich nicht die Rede sein. Wir wollten damit nur verdeutlichen, daß man das Zugriffskriterium auch als Schlüssel sehen kann.
Auf eine ISAM-Datei wird grundsätzlich über einen (echten) Schlüssel zugegriffen. Hier reicht nicht das Wissen um die Hausnummer, sondern Sie müssen die Bewohner schon sehr genau kennen.
Was heißt das?
Der Schlüssel zum Satz einer ISAM-Datei ist regelmäßig Bestandteil des Satzes selbst oder ein logischer Auszug davon.
Nehmen wir wieder unsere Adressdatei. Wir haben in der REL-Beschreibung gesagt, daß wir mit einem Verzeichnis, das den Zusammenhang zwischen Satznummer und Nachnamen herstellt, auch den vollständigen Eintrag erfahren können.
Bei ISAM könnte der Name selbst (oder die Postleitzahl oder ...) das Zugriffskriterium sein.
Wie geht das vor sich?
Beim Einrichten einer ISAM-Datei müssen Sie nur außer der Satzlänge auch die Position und die Länge des Schlüsselkriteriums innerhalb des Satzes bestimmen. Es ist klar, daß Sie die Satznummer nicht mehr benötigen, denn die Position des Eintrages innerhalb der gesamten Datei wird aus den zu schreibenden Daten selbst entnommen.

Nun aber kommt der Clou:
Wir kehren noch einmal kurz zu den REL-Files zurück.

Angenommen, Sie gewinnen die Satznummer aus einer rechnerischen Behandlung des Satzinhaltes, beispielsweise aus asc(erstes Zeichen). Rein theoretisch kann hier ein Wert von 0 bis 255 herauskommen, den Sie nun als Satznummer nehmen. Sicher haben Sie so eine gewisse Beziehung zwischen Satznummer und Satzinhalt hergestellt, jedoch wird diese Beziehung Sie spätestens dann enttäuschen, wenn Sie einen neuen Satz einspeichern wollen, der mit dem gleichen Zeichen beginnt. Der alte Satzinhalt wird nämlich durch den neuen überschrieben, da auch die resultierende Satznummer die gleiche ist.
Anders bei ISAM. Sicher kann auch hier innerhalb des als Schlüssel definierten Feldes die gleiche Zeichenkombination wiederholt auftauchen, wenn auch nicht so oft, wie in unserem Beispiel. Tritt ein solcher Fall auf, wird der neue Satz einfach zwischen den schon vorhandenen Satz mit demselben Schlüsselfeldinhalt und dem Satz mit dem nächsthöheren Schlüssel eingeschoben.
Versuchen Sie das einmal bei einer REL-Datei!

Auch beim fortlaufenden Lesen gibt es einen kleinen Unterschied:
Es wird nicht einfach der Satz mit der nächsthöheren Nummer zur Vefügung gestellt, sondern der Satz mit dem logisch nächsthöheren Schlüsselfeldinhalt.
Was bedeutet das konkret?
Wieder zurück zur Adressdatei. Sie haben mit ISAM eine Adressensammlung aufgebaut, wobei Sie als Schlüssel die ersten zehn Stellen des Nachnamens definiert haben. Man sehe und staune: beim fortlaufenden Lesen der Sätze werden diese in alphanumerischer Reihenfolge der Nachnamen herausgegeben. In diesen zehn Stellen übereinstimmende Sätze werden unmittelbar aufeinanderfolgend ausgegeben.
Also eine äußerst praktische Angelegenheit, bei der die Ausgabe gleich sortiert erfolgt.
Sie können sich sicher vorstellen, daß selbstverständlich für diese logische Kettung der Schlüssel ein gewaltiger Verwaltungsaufwand in der Floppy getrieben werden muß. Deshalb ist diese Möglichkeit im Hobbycomputerbereich auch

nicht sehr verbreitet.

Aber auch von ISAM gibt es noch eine Steigerung:

Das Zauberwort heißt VSAM.
????????????????
Vertical Sequential Access Method.
Ohne in aller Ausführlichkeit darauf einzugehen, sei hier soviel dazu gesagt, daß Sie es bei VSAM nicht nur mit einem Schlüssel, sondern gleich mit mehreren zu tun haben, von denen jeder als Zugriffskriterium hergenommen werden kann.
Sie haben in einer mit VSAM aufgebauten Adressdatei den Nachnamen als erstes Schlüsselfeld definiert und die Postleitzahl als zweites.
Fortlaufendes Lesen bringt Ihnen, wie gehabt, die Einträge in Alphabetischer Reihenfolge der Nachnamen. Ein fortlaufender Zugriff über den zweiten Schlüssel stellt Ihnen jedoch die Sätze in numerischer Reihenfolge der Postleitzahlen zur Verfügung. Ebensogut hätten Sie noch weitere Schlüssel definieren können wie z.B. Straßennamen und Telefonnummer, die dann allesamt als Zugriffsargument zur Verfügung ständen.
Zur Information sei noch angemerkt, daß es für den CBM8032 ein solches Zugriffssystem unter dem Namen SUPERKRAM gibt.
Nach diesem Ausflug in die schöne Welt der gehobenen Datenverarbeitung wollen wir uns wieder auf den Boden des Machbaren begeben.
Lesen Sie im nächsten Abschnitt, welcher Dateiverwaltungskomfort Ihnen auch mit der VC1541 zur Verfügung stehen kann. 'Kann' deshalb, weil doch einiger Programmieraufwand damit verbunden ist. Aber wie Sie uns kennen, haben wir auch dafür fertige Routinen.

2.3.1.4 QUISAM

Wir möchten Ihnen an dieser Stelle ein Datenverwaltungssystem vorstellen, das wir uns ausgeheckt haben und in dem wir der Versuchung erlegen sind, die Vorteile der Dateiarten SEQ, REL zu vereinigen, ohne nach Möglichkeit die spezifischen Mängel einer jeden, die einer wirklich universellen Anwendung im Wege stehen, mit zu übernehmen, um so zu einer Art ISAM-Datei zu kommen.
Daher haben wir auch den Namen:
QUISAM soll nämlich, wie Sie sich sicher schon gedacht haben, QUasi-ISAM bedeuten. Im Folgenden wird sehr ausführlich beschrieben, wie QUISAM funktioniert. Es empfiehlt sich in jedem Fall, nicht nur die Routinen abzutippen, sondern auch die einzelenen Abschnitte sorgfältig durchzuarbeiten. Sie erhalten so einen erheblich tieferen Einblick nicht nur in QUISAM, sondern in das wichtige Thema Dateiverwaltung überhaupt.
Stellen wir die Vor- und Nachteile der zuvor erwähnten Grunddateiarten REL und SEQ noch einmal kurz heraus:

REL Vorteil:

Schneller Zugriff auf beliegige Sätze.
Änderungs- und erweiterungsfreundlich.

Nachteil:

In der Regel ist die Datei größer, als den tatsächlich belegten Sätzen entspricht.
Datensätze können nicht im nachhinein vergrößert werden.
Zugriffskriterium ist nur die Satznummer.

SEQ Vorteil:

Schnelle Speicherung großer Mengen fortlaufend anfallender Daten.
Dynamisch erweiterbar, d.h. Dateigröße wächst kontinuierlich mit den anfallenden Daten.

Nachteil:

Kein wahlfreier Zugriff auf beliebige Daten innerhalb der Datei.
Änderungsunfreundlich.

Wie also soll nun eine Dateiarchitektur aussehen, die möglichst vielen Anforderungen genügt?

1. Auf jeden Fall wahlfreier Zugriff auf beliebige Sätze anhand einer Satznummer.
2. Zugriffsmöglichkeit über einen echten Schlüssel.
3. Dynamische Erweiterung der Datei mit wachsendem Datenanfall.
4. Dynamische Erweiterung auch eines einzelnen Satzes.
5. Bei allem Komfort eine vernünftige Zugriffszeit.
6. Einfache Anwendung von Basic aus.

Wie haben wir das realisiert?
Wir bleiben beim Beispiel der Adressdatei, um die folgenden Ausführungen zu illustrieren.
Zunächst geht es darum, einen sog. Stammsatz anzulegen. Das ist ein Satz, in dem die wesentlichen Merkmale des z.B. Kunden untergebracht sind. Dazu zählt außer dem Namen auch die Anschrift usw., also alle Angaben, die zur Adressierung eines Briefes benötigt werden.
Wie aber vergeben wir die Satznummer?
Eine einfache aufsteigende Numerierung ist wegen der Notwendigkeit, nebenher eine Zuordnungsliste zu führen, unpraktisch. Also ermitteln wir aus dem Bereich des Satzes, der den Nachnamen enthält, eine Zahl. Wie wir es anstellen, eine möglichst originelle, also einmalige Zahl zu finden, zeigen wir Ihnen im nächsten Abschnitt.

2.3.1.4.1 DER HASHCODE

Unter Hashcode ist eine Zahl zu verstehen, die aus einer Information gewonnen wird und den zielsicheren Zugriff auf diese Information ermöglichen soll. Was bedeutet das?
Das ist an einem einfachen Beispiel schnell erklärt:
Nehmen wir einmal an, wir hätten alle Buchstaben des Alphabets in einer REL-Datei derart gespeichert, daß je ein Buchstabe einen Satz belegt. Um jetzt jeden Buchstaben mit einem Zugriff hervorholen zu können, müssen wir wissen, wo er steht. Wir haben deshalb beim Speichern die Satznummer aus dem Buchstaben selbst berechnet, nämlich mit der Formel

```
r=asc(r$)
```

wobei r$ unser Zeichen enthält. Wir bekommen für r eine Zahl zwischen 65 und 90 heraus.
Wollen wir nun den Buchstaben wieder hervorholen, wenden wir die gleiche Formel an. Wenn wir mit dieser Zahl auf unsere Datei zugreifen, erhalten wir genau den Buchstaben wieder zurück. Natürlich ist ein Vorgehen in dieser Form unsinnig, da Suchargument und Information identisch sind. Sicher ließe sich das Verfahren auch auf eine Kombination von zwei Buchstaben ausdehnen. Die Satznummer würde dabei aber schon vierstellig. Bei drei Buchstaben würden wir schon den Bereich der gültigen Satznummern verlassen. Außerdem hat das Verfahren den Nachteil, daß der größte Teil der Datei ungenutzt bliebe, da schon beim ersten Beispiel mit nur einem Buchstaben die Sätze 1 bis 64 frei bleiben.
Es geht also im Grunde darum, den Bereich optimal zu nutzen, bei größtmöglicher Originalität der Satznummern.
Wie wäre es, wenn wir aus der Formel nur den niedrigstwertigen Zahlenteil entnehmen, also etwa so:

```
r=val(right$(str$(asc(r$)),1))
```

Für r$="a" erhalten wir r=5. Das trifft leider auch für r$="k" zu.
Wir sehen, daß das Verfahren für einen Buchstaben ungeeignet ist. Anders sieht es aus, wenn wir vier Buchstaben hernehmen. Wir erhalten dann eine Zahl zwischen 0 und 9999. r$="fpzf" beispielsweise wird 0 ergeben, und z.B. r$="eoyo" ergibt 9999. Das trifft auch für jede beliebige Kombination

aus den jeweiligen Buchstaben zu, aber da wir ja gewöhnlich mit 'echten' Daten, also Nachnamen arbeiten, ist die Wahrscheinlichkeit einer originellen Zahl ungeheuer hoch.
Übrigens, falls Sie die obige Formel mit mehr als einem Buchstaben versuchen sollten, bekommen Sie immer nur eine zweistellige Zahl zurück. Wie eine mehrstellige Zahl gebildet wird, werden wir Ihnen noch zeigen.
In der Regel werden zur Bildung eines Hashcodes fünf Zeichen der Information herangezogen. Welche fünf Stellen, das muß der Anwender entscheiden, da nur er wissen kann, welches Feld eines Datensatzes sich am meisten vom gleichen Feld eines anderen Satzes unterscheidet. Im Falle der Adressdatei wäre es natürlich wenig sinnvoll, das Feld für die Anrede herzunehmen.
Nun tritt ein weiteres Problem auf:
Bei einem fünfstelligen Schlüsselfeld erhalten Sie auch eine fünfstellige Zahl, die jedoch die erlaubten Grenzen für die Satznummer sprengen könnte. Selbst wenn das möglich wäre, ist es dennoch nicht sinnvoll, die Datei wirklich so groß zu machen, denn wer hat schon 99999 Adressen zu verwalten.
Es gilt also, eine Zahl zu ermitteln, die sowohl originell ist, als auch den erlaubten Bereich nicht nur nicht verläßt, sondern darüberhinaus sich sogar nur in einer Größe bewegt, die auch der tatsächlich anfallenden Menge von Adressen entspricht.
Was nun?
Nehmen wir an, bei uns würden etwa 100 Adressen anfallen. Damit ist schon klar, daß die Zahl sich im Bereich 1-100 bewegen muß.
Trotzdem ermitteln wir zunächst eine fünfstellige Ziffer aus den ersten fünf Zeichen des Nachnamens.
Jetzt geht es los:
Sicher kennen Sie die Funktion a=rnd(x). Hierbei kann a einen Wert im Bereich 0<a<1 annehmen. Wollen Sie aber eine Zufallszahl zwischen 1 und 49 (für den Lottospieler), ist die Formel

```
a%=rnd(x)*50
```

anzuwenden.
Wir brauchen eine Zahl zwischen 1 und 100. Also müssen wir

zuerst aus unserer fünfstelligen Zahl eine Zahl 0<>1 machen, wobei die Anzahl der signifikanten Stellen möglichst groß sein soll, damit wir unseren Bereich ausfüllen können.
Würden Sie beispielsweise die aus den fünf Zeichen ermittelte Zahl einfach durch 100 teilen, so wären die resultierenden Nachkommastellen 'langweilig' und damit auch die Satznummer.
Ein Beispiel:
Sie haben die Zahl 75460 ermittelt. Diese teilen wir durch 100. Heraus kommt 754.6. Sie trennen die Nachkommastellen ab und multiplizieren diese wieder mit 100. Heraus kommt 60, also eine zwar brauchbare Satzzahl, die jedoch den Nachteil hat, daß sie viel zu oft vorkommen kann, nämlich bei allen Zahlen, die mit ...60 enden. Teilen wir aber 75460 durch 101, so ergibt das 747,128713. Das sieht schon ganz anders aus. Trennen Sie die Nachkommastellen ab und multiplizieren Sie diese mit 100. Entnehmen Sie dem Ergebnis den Integer-Teil, so ergibt das die gesuchte Satznummer 12.
Nach dieser Formel bekäme der Herr Schmitz die Nummer 5, der Herr Schulz die Nummer 29 und Herr Müller 83.
Sie wollen wissen, wie wir ausgerechnet auf 101 als Teiler gekommen sind?
Nun, 101 ist eine Primzahl (eine Zahl, die sich nur durch sich selbst und durch 1 geteilt eine ganze Zahl ergibt), und Primzahlen haben als Teiler die Eigenschaft, daß sie 'wüste' Zahlen mit vielen Nachkommastellen erzeugen.
Der langen Rede kurzer Sinn:
Als Faustregel soll gelten, daß als Teiler eine Primzahl genommen wird, die nächstgrößer als die beabsichtigte Satzmenge für die Datei ist. Für eine Datei mit 1000 Sätzen wäre der Teiler 1009.

Natürlich haben wir Ihnen das etwas mühsame Geschäft des Hashcode-Bildens wieder mit einer kleinen Routine abgenommen.
Alles, was sie zu ihrem Ablauf benötigt, ist der vollständige Datensatz, die Lage des Schlüsselkriteriums (beim wievielten Zeichen des Satzes beginnt der Schlüssel?), die Länge des Schlüssels, die Primzahl und die Dateigröße.

Da diese Parameter (mit Ausnahme des Datensatzes) für die gesamte Behandlungsdauer der Datei festliegen, brauchen sie auch nur einmal definiert zu werden.

Hier nun das Programmchen:

```
58800 rem
58801 rem*************** hashcode bilden ***************
58802 rem
58820 hc$=mid$(qd$,qk%,qh%)+"     "
58840 hc=0:forl1=0to4
58860 hc=hc+val(right$(str$(asc(mid$(hc$,l1+1,1))),1))*10^l1
58880 nextl1
58900 hc=int((hc/qp-int(hc/qp))*qb)+2
58920 return
```

Die Variablen sind wie folgt belegt:

qd$ muß den vollständigen Datensatz enthalten.

qk% enthält die Lage des Schlüsselkriteriums.

qh% enthält die Länge des Schlüssels. Damit keine Mißverständnisse aufkommen: der Hashcode wird in dieser Routine immer über fünf Zeichen gebildet, unabhängig vom Wert in qh%. Die echte Länge erhält jedoch Bedeutung im Zusammenhang mit unserer QUISAM-Datei. In unserem Beispiel soll der Schlüssel die gesamte Länge des Nachnamenfeldes umfassen, nämlich 12 mit der Lage qk%=16.

qb sollte die maximale Dateigröße (wieviele Sätze soll sie höchstens umfassen), und

qp die besagte Primzahl enthalten.

hc$ enthält die Zeichenfolge, die zur Bildung der Satznummer hergenommen wird.

hc endlich enthält die fertige SatznummeR.

Es folgt der Kommentar.

58820 hc$ wird aus qd$ mit den Parametern qk% und qh% isoliert und auf fünf Zeichen aufgefüllt, falls qh% kleiner als fünf ist.

58840-
58880 In dieser Schleife wird die fünfstellige Zahl gebildet. So kriminell die Formel auch aussehen mag, so einfach ist sie:
Von links beginnend, wird von jedem Zeichen der asc-Wert gebildet, dessen rechte Ziffer abgetrennt und mit 10, potenziert mit der Laufvariablen l1, multipliziert und diese Zahl fortlaufend auf hc addiert.
Das Ergebnis ist eine bis zu fünfstellige Zahl, deren Reihenfolge der Ziffern umgekehrt ist, wie eigentlich nach der Zeichenfolge zuerwarten wäre. Das liegt einfach daran, daß wir zwar links im Schlüsselfeld beginnen, jedoch mit der kleinsten Potenz von 10 als Multiplikator anfangen. Der Brauchbarkeit tut das keinen Abbruch.

58900 Da als Hashcode ohne weiteres die Zahl Null auftauchen kann, diese jedoch als Satznummer nicht erlaubt ist, wird hc gleich um zwei erhöht, d.h. Ihr erster Block beginnt bei der Satznummer zwei. Das haben wir deshalb so eingerichetet, weil Satz eins für die Verwaltung der QUISAM-Datei benötigt wird. Hier werden z.B. beschreibende Parameter wir qb, qp, qk%, qh% usw. abgelegt, was Sie später der Mühe enthebt, beim Eröffnen einer QUISAM-Datei diese Werte aus dem Gedächtnis zu versorgen, was Ihnen bei mehreren Dateien unterschiedlichen Formats schwerfallen dürfte.

Mit dieser Routine und den Unterprogrammen zur Handhabung von REL-Files können Sie schon eine einfache Adressverwaltung aufbauen. Wenn Sie zur Satzsuche denselben Algorhitmus wie beim Schreiben benutzen, werden Sie den vollständigen Satz durch einfaches Angeben des Nachnamens frei Haus geliefert bekommen. Wenn Sie zur Suche nur den Nachnamen in qd$ an die Routine übergeben, müssen Sie natürlich qk%=1 setzen.

Spielen Sie ruhig ein wenig damit. Wir sind sicher, daß Sie Ihren Spaß daran haben werden. Gelegentlich stoßen Sie darauf, daß für unterschiedliche Namen dieselbe Satznummer berechnet wurde. Das ist normal, aber aus dieser Verlegenheit werden wir Ihnen im nächsten Abschnitt heraushelfen. Ein Tip: Je größer Sie die Datei wählen, umso seltener kommt es vor.

2.3.1.4.2 DATEIGRöSSE

Ein äußerst wichtiger Punkt bei der Anlage von Dateien sind Betrachtungen über ihre Größe. Wir wollen überlegen, was dazu geführt haben kann, daß bei einem Vorgehen, wie im vorigen Abschnitt beschrieben, die gleiche Satznummer unter Umständen auch mehrere Male auftreten kann.
Gewiß trägt dazu bei, daß eine aus der Information gewonnene zunächst fünfstellige Zahl auf eine Zahl mit weniger Stellen zurechtgestutzt wird, um eine Satznummer entsprechend Ihrer Dateigröße zu erhalten. Da dabei notwendigerweise Stellen abgeschnitten werden, kann der Fall auftreten, daß die übrigen Stellen bei ursprünglich unterschiedlichen Schlüsselbegriffen gleich sind. Daher auch der Tip, die Datei zu vergrößern, damit die Satznummern über einen größeren Bereich gespreizt sind.
Das hilft sicher auch anfangs, jedoch handeln Sie sich einen gravierenden Nachteil ein: Es bleiben mehr Sätze in der Datei frei, was die reine Platzverschwendung bedeutet.
Noch etwas kommt hinzu: Da von den Dezimaläquivalenten der Schlüsselzeichen nur deren letzte Stelle hergenommen wird, tritt auch hier schon ab und zu eine Übereinstimmung der resultierenden Zahl bei unterschiedlichen Schlüsselbegriffen auf.
Was also ist zu tun?
Eine Möglichkeit wäre die, daß Sie, wenn festgestellt wird, daß eine zu beschreibende Satzposition bereits belegt ist, einfach die nächsthöhere nehmen und so fort, bis Sie einen freien Satz finden.
Das macht aber auch das Wiederauffinden schwieriger, weil

Sie jetzt damit rechnen müssen, daß Sie beim Zugriff mit dem Hashcode nicht die Information bekommen, die sie eigentlich wollten.
Sie müssen dennoch zunächst den Satz lesen, dessen Schlüsselbegriff mit dem gesuchten vergleichen und bei Ungleichheit den nächsthöheren Satz lesen, bis Sie entweder den Schlüssel gefunden haben oder bis Sie auf einen freien Satz stoßen, was bedeutet, daß der gesuchte Begriff nicht vorhanden ist.
Ein solches Vorgehen ergibt immer noch eine respektable Zugriffsgeschwindigkeit, gemessen an der Suche aus einem SEQ-File. Es ist je nach Belegungsdichte der Datei mit bis zu sechs Zugriffen zu rechnen, bis der gesuchte Satz gefunden ist, was ganz schön schnell geht, wenn die Floppy einmal in Schwung ist.
Ein Problem ist aber damit immer noch nicht erschlagen, nämlich das der unnötigerweise freien Sätze.

Folgende Überlegung soll Sie auf die richtige Spur führen:
Da Sie ja ohnehin mit mehreren Zugriffen bis zum Erfolg rechnen müssen, warum die ganze Datei nicht gleich kleiner machen, und zwar so klein, daß sie auf jeden Fall möglichst voll wird, also nur noch wenige freie Sätze aufweist.
Ja, werden Sie einwenden, aber wohin mit den Sätzen gleichen Hashcodes?
Ganz einfach: Für diese wird eine zweite Datei angelegt, in die sie der Reihe nach kommen.
Diese Überlegung hat viel für sich, denn es ist absolut gleichgültig, ob Sie nach dem ersten Mißgriff von da an in der gleichen Datei Satz für Satz absuchen müssen, bis Sie den gesuchten Schlüssel gefunden haben, oder in einer anderen.
Einleuchtend, nicht wahr?
Es gibt dabei aber noch einen weiteren Vorteil: Diese Überlaufdatei lassen Sie immer gerade nur so groß werden, wie es der tatsächlichen Zahl der Einträge entspricht, d.h. immer, wenn ein Überlaufsatz (ein Satz, der in der Primärdatei keinen Platz mehr findet, weil die Position schon besetz ist) auftritt, wird die Sekundärdatei gerade um

einen Satz vergrößert.
Nun haben Sie alles, was Sie brauchen: Eine ziemlich kompakte Primärdatei, in der Sie auch ab und zu sofort fündig werden, und eine dynamisch wachsende Sekundärdatei, in der der Rest steht.
Natürlich sollten Sie, wenn Sie sich bei der Kalkulation der anfallenden Daten verschätzt haben (das merken Sie an erhöhten Zugriffszeiten), die Primärdatei vergrößern, wobei Sie aber unbedingt die alten Daten unter Zugrundelegung der nun anderen Formel zur Hashcodeermittlung in die neue, größere Datei überführen. Es geht also auf keinen Fall, daß Sie einfach die bestehende Datei vergrößern und neue Einträge mit einer anderen Formel vornehmen. Sie würden die alten nicht mehr wiederfinden, da aufgrund der neuen Formel auch andere Satznummern auftreten.

Wie Sie sicher schon gemerkt haben, verbirgt sich in diesen Überlegungen in grober Fehler.
Gehen wir einmal davon aus, daß die Sekundärdatei noch völlig leer ist. Nun wollen Sie einen Eintrag machen, finden den Platz in der Primärdatei belegt und schreiben ihn deshalb in die Sekundärdatei.
Später wollen Sie diesen Satz wieder lesen. Sie ermitteln die Satznummer und greifen damit zunächst auf die Primärdatei zu.
Ein Schlüsselvergleich zeigt Ihnen, daß Sie den falschen Satz erwischt haben. Also lesen Sie den ersten Satz aus der Sekundärdatei. Aha, dieser ist es.
Alles bestens, oder?
Sie wollen einen weiteren Satz mit einem total anderen Schlüssel und anderer Satznummer in die Primärdatei schreiben, finden jedoch auc diese Position besetzt. Also hinein damit in die Sekundärdatei.
So geht es noch einige Male.
Jetzt versuchen Sie einmal, den zuletzt geschriebenen Satz wieder hervorzuholen. Wenn Sie zuvor hundert Einträge vorgenommen haben, die allesamt in der Sekundärdatei gelandet sind, dürfen Sie auch zunächst hundert Sätze überlesen, bevor Sie fündig werden.

Tatsächlich, das stimmt, aber es war doch vorhin die Rede von maximal sechs Zugriffen!
Stimmt auch, denn damit es nicht allzu einfach bleibt, bedienen wir uns nun eines kleinen Tricks, der es Ihnen ermöglicht, die Sekundärdatei erst ab dem Eintrag lesen zu müssen, der als erster mit der Satznummer in der Primärdatei korrespondiert, d.h. der Eintrag, der beim erstmaligen Überlauf der in Rede stehenden Satznummer aufgetreten ist.
Das klingt schon besser, aber wie soll das gehen?
Ganz einfach: Sie reservieren in jedem Satz der Primärdatei einen Platz, auf dem beim Auftreten des ersten Überlaufs für diesen Satz einfach die Satznummer des Folgeblockes (das wird in der Regel der erste freie Satz der Sekundärdatei sein) eingetragen wird.
Suchen Sie nun einen Begriff, so entnehmen Sie bei einem Fehlschlag in der Primärdatei einfach aus der bestimmten Stelle die dort eingetragene Zahl und greifen damit auf die Sekundärdatei zu. Damit haben Sie sich eine gewaltige Sucherei erspart.
Um das Maß nun restlos voll zu machen, erhält jeder Satz der Sekundärdatei auch noch eine Kennfeld, aus dem hervorgeht, wo ein evtl. weiterer Folgeblock in derselben Datei zu finden ist, denn es muß durchaus nicht immer der nächste sein, wenn inzwischen weitere Überläufe anderer Primärsätze aufgetreten sind.
Als Kennzeichen, daß keine weiteren Überlaufsätze für den gleichen Hashcode existieren, tragen Sie in das Kennfeld eine Null ein, bei deren Auftreten Sie die Suche beenden können, denn dann ist der verlangte Eintrag nicht vorhanden.

Wir wissen, daß sich das alles furchtbar trocken anhört.
Deshalb finden Sie in Abb. 2.3-1 ein Schema, wie Sie sich das vorzustellen haben.

PRIMÄRDATEI			SEKUNDÄRDATEI		
Aus dem Hashcode gewonnene Satznummer	Kennfeld	Daten	Satznummer	Kennfeld	DatEn
:	:		:	:	
:	:	:	:	:	:
2	4	Schmitz1	2	3	Müller2
3			3	9	Müller3
4	5	Meier1	4	6	Schmitz2
5			5	0	Meier2
6			6	0	Schmitz3
7	7	Schultz1	7	8	Schultz2
8			8	0	Schultz3
9					
10	2	Müller1			

Abb. 2.3-1

Sie können jetzt deutlich erkennen, daß es in der Primärdatei wohl noch einige Lücken gibt, die Sekundärdatei jedoch gerammelt voll ist, da sie ja erst bei tatsächlichem Bedarf erweitert wird.
In beiden Dateien ist der erste Block nicht belegt. Den brauchen wir später, um ein wenig Verwaltung treiben zu können.

Jetzt werden wir dem i noch das Tüpfelchen aufsetzen:
Ab und zu kann die Notwendigkeit auftreten, einem Eintrag noch einige Anmerkungen hinzuzufügen. Sollte es sich bei den Adressen um Ihre Kunden handeln, würden Sie sicher gerne noch weitere Informationen hinterlegen, etwa in der Art, ob es sich um einen unsicheren Kantonisten handelt oder ob er immer Vorkasse leistet oder ob er Rolls Royce fährt usw.
Es wäre natürlich Unfug, wenn Sie solche Einträge bereits bei der Satzgröße berücksichtigten, da derartige Einträge sicher nicht bei allen Kunden erfolgen und auch die zu

erwartende Länge nicht vorhersehbar ist. Die Platzverschwendung wüchse ins Gigantische.
Was liegt nach den bisherigen Erfahrungen näher, als dafür eine dritte Datei mit geringer Satzlänge anzulegen. Der Dateiaufbau hat das gleiche Schema wie die Sekundärdatei, also Erweiterung nur bei Bedarf. Die Satzlänge ist so zu wählen, daß Bemerkungen durchschnittlicher Länge (wie groß, das müssen Sie wissen) gerade Platz haben.
Auch der Zugriff erfolgt auf die gleiche Weise, wie der Zugriff auf die Sekundärdatei. Dazu müssen Sie aber in den Stammsätzen ein weiteres Kennfeld reservieren, aus dem ersichtlich ist, ob (Kennfeld 0) und wenn ja, wo die Bemerkungen in der dritten Datei stehen.
Die Verkettung der zu einem Stammsatz gehörenden Bemerkungen untereinander erfolgt wiederum so wie bei der Sekundärdatei. Sollte ein Eintrag länger als die definierte Satzlänge werden, hängen Sie einfach einen Folgesatz an.
Die Verkettung hat außerdem den Vorteil, daß Bemerkungen zu einem bestimmten Stammsatz auch noch Jahre später angehängt werden können. Sie finden sie alle wieder.
Die neue und nunmehr endgültige Gesamtstruktur finden Sie in der Abb. 2.3-2 illustriert.
Sie werden zugeben, daß es universeller und komfortabler kaum geht.
Der Schritt zu einer selbstverwaltenden QUISAM-Datei ist nun nur noch ein kleiner.
Das erste Programm, welches wir Ihnen in diesem Zusammenhang vorstellen wollen, ist das Einrichten einer QUISAM-Anwendung. Das erfordert auch schon die größte Aktivität Ihrerseits, denn von da an geht alles (fast) automatisch.

PRIMÄRDATEI

Satz	Folge.se	Folge.ex	lfd.Nr.	Daten
2	2	2	0	Müller1
4	0	3	0	Schultz1
15	4	8	0	Hansen1

SEKUNDÄRDATEI

Satz	Folge.se	Folge.ex	lfd.Nr.	Daten
1	6			
2	3	5	1	Meier1
3	0	6	2	Meier2
4	5	9	1	Hansen2
5	0	10	2	Hansen3

BEMERKUNGEN

Satz	Folge.ex	Daten
1	12	
2	0	Tel. 69704
3	4	hübsche Tochter
4	0	zum Essen einladen
5	0	schuldet DM 100
6	7	am 22.12. besuchen
7	0	Blumen mitnehmen
8	0	Aschenputtel
9	0	Kellner im Ochsen
10	11	am 20.1. Geburtstag
11	0	nicht vergessen!

Abb. 2.3-2

Das Programm sieht so aus:

```
59000 rem
59001 rem************* anlegen quisam *******************
59002 rem
59020 printchr$(147)"anlegen einer quisam-datei"
59030 print
59040 input"dateiname (max. 8 zeichen)";qn$
59050 print:iflen(qn$)>8then59040
59060 input"wieviele saetze ca.";qb
59070 print
59080 input"laenge stammsatz (min.4, max.249)";ql%
59090 print:ifql%<4orql%>249then59080
59100 input"laenge anhangsatz (min.2, max.252)";qx%
59110 print:ifqx%<2orqx%>252then59100
59120 input"startposition schluessel";qk%
59130 print
59140 input"laenge schluessel";qh%
59150 if(qk%+qh%-1)>ql%then59120
59160 qb=int(qb/6)+1:qp=qb+1
59180 ifqpand1then59200
59190 qp=qp+1:qx=2:qs=2
59200 open15,8,15
59210 l1=3
59220 ifqp/l1=int(qp/l1)thenqp=qp+2:goto59210
59240 l2=int(qp/l1):l1=l1+1:ifl1<l2then59220
59260 dn$=qn$+".pr":r=qb+2:rl%=ql%+6:gosub63800
59280 r$=chr$(ql%)+chr$(qx%)+chr$(qk%)+chr$(qh%)
59300 rh%=qb/256:rl%=qb-rh%*256
59320 r$=r$+chr$(rh%)+chr$(rl%)
59340 rh%=qp/256:rl%=qp-256*rh%
59360 r$=r$+chr$(rh%)+chr$(rl%)
59380 r=1:gosub63000
59400 r$=chr$(0)+chr$(0)+chr$(0):r$=r$+r$
59460 print#15,"p"chr$(6)chr$(2)chr$(0)chr$(1)
59480 forl1=2toqb+1
59500 print#6,r$:nextl1:close6
59520 dn$=qn$+".se":r=1:rl%=ql%+6:gosub63800
```

```
59540 r$=chr$(0)+chr$(2):gosub63000
59560 dn$=qn$+".ex":rl%=qx%+3:gosub63800
59580 gosub63000:return
```

In bewährter Form wollen wir zuerst die Variablen erläutern:

qn$ Name der QUISAM-Datei. Die Begrenzung auf acht Stellen verlangen wir wegen der vom Programm automatisch angefügten Anhänge an den Namen: ".pr" für die Primärdatei, ".se" für die Sekundärdatei und ".Ex" für die Bemerkungen.

qb von Ihnen geschätzte Anzahl der benötigten Sätze.

ql% Länge eines Stammsatzes. Bei einer Adressdatei der von uns vorgeschlagenen Form wäre das 67.

qx% Länge eines Bemerkungssatzes. Das wissen Sie sicher besser als wir. 20 wäre ein guter Wert.

qk% Diese Variable kennen Sie schon aus der Hashcoderoutine. Hiermit geben Sie an, an welcher Position des Satzes der Schlüsselbegriff zu finden ist. In unserem Adressformat ist das 16.

qh% ist die Länge des Schlüssels, bei uns 12.

qp ist der aus der Satzanzahl resultierende Primfaktor und wird von der Routine selbst berechnet.

Und so funktioniert es:

59020-
59150 Die beschreibenden Parameter werden im Dialog von Ihnen abgefordert.

59160 Wir unterstellen einfach, daß Sie sich mit einem bis zu sechsfachen Suchzugriff zufriedengeben. Deshalb teilen wir zur Ermittlung der Größe für die Primärdatei einfach die von Ihnen angegebene Satzzahl durch sechs. Sollten Ihnen mit der Zeit die Zugriffe zu lange dauern, passen Sie den Teiler einfach Ihrem Geschmack an.

Die so gewonnene Zahl wird um eins erhöht, da wir ja auch noch einen Satz für die Verwaltung der Datei

brauchen.

59180-
59240 Hier wird die Primzahl qp berechnet. Es wird, grob gesagt, einfach probiert, ob sich die Zahl (wir beginnen qb+1) durch eine andere Zahl (hier beginnen wir mit 2) ganzzahlig teilen läßt. Ist das der Fall, so ist es keine Primzahl und wir probieren es mit der nächstgrößeren Zahl. Der Versuch der Teilung wird solange fortgesetzt, bis ein Limit erreicht ist, bei dem wir sagen können, daß die nun folgenden Zahlen als Teiler auch nicht mehr infrage kommen.

59260 Die Primärdatei wird in der vollen Länge angelegt.

59280-
59380 Der erste Satz der Primärdatei wird mit den beschreibenden Parametern geladen. Das enthebt Sie später der Mühe, diese Variablen im Gedächtnis zu behalten.

59400-
59500 Jeder Satz der Primärdatei wird mit sechs binären Nullen beschrieben. Das wird später gebraucht, um einen leeren Satz zu erkennen.

59520-
59580 Die Sekundärdatei und die Anhangdatei werden je mit dem Satz 1 angelegt. In diesem Satz wird die Nummer des ersten freien Satzes (merken Sie, wo es lang geht?), in diesem Falle 2, hinterlegt.

Wir fahren am besten gleich mit dem Programm zur Eröffnung einer bestehenden QUISAM-Datei fort, damit Sie sehen können, wozu die ersten Sätze jeder Datei nütze waren.

```
58000 rem
58001 rem***************** oeffnen quisam *************
58002 rem
58010 close6:close15:open15,8,15:rz$=""
58020 dn$=qn$+".pr":r=1:rc%=8:gosub63500
58040 ql%=asc(left$(r$,1))
58060 qx%=asc(mid$(r$,2,1))
58080 qk%=asc(mid$(r$,3,1))
58100 qh%=asc(mid$(r$,4,1))
58120 qb=asc(mid$(r$,5,1))*256+asc(mid$(r$,6,1))
58140 qp=asc(mid$(r$,7,1))*256+asc(right$(r$,1))
58160 dn$=qn$+".se":rc%=2:gosub63500
58180 qs=asc(left$(r$,1))*256+asc(right$(r$,1))
58200 dn$=qn$+".ex":gosub63500
58220 qx=asc(left$(r$,1))*256+asc(right$(r$,1))
58240 return
58300 rem
58301 rem************* schliessen quisam **********
58302 rem
58320 close6:return
```

Die Variablen:

qn$ ist die einzige Variable, mit der diese Routine versorgt werden muß. Sie soll natürlich den Dateinamen enthalten.

qs kommt neu hinzu. Hierin wird die Nummer des ersten freien Satzes der Sekundärdatei hinterlegt. Diese Zahl stammt aus deren Satz 1.

qx hat die gleiche Bedeutung wie qs, jedoch auf die Bemerkungsdatei bezogen.

Nun der Kommentar:

58010-
58140 Die ersten acht Bytes des ersten Satzes werden aus der Primärdatei gelesen. Sie werden den beschreibenden Variablen zugeordnet, die ja beim Anlegen der Datei dort hinterlegt wurden. So ist sichergestellt, daß zur

Bearbeitung einer QUISAM-Datei immer auch die Parameter hergenommen werden, die auch für den Aufbau maßgebend waren.

58160-
58180 Die Nummer des ersten freien Satzes wird aus der Sekundärdatei geholt und qs zugeordnet.

58200-
58220 Wie oben, jedoch für Bemerkungsdatei und qx.

58300-
58320 Der Datenkanal wird einfach hier an zentraler Stelle geschlossen, damit Sie auch nur hier zu ändern brauchen, falls Sie bei den REL-Zugriffen die lfn geändert haben.

So, nun haben Sie eigentlich schon alles, was Sie brauchen, um die bisher besprochenen Dinge durchführen zu können.
Ihre Arbeit besteht nun nur noch darin, die bisher vorgestellten Routinen, angefangen von der Bildschirmmaske bis hierher, sinnvoll miteinander zu verknüpfen und mit den benötigten Variablen zu versorgen.
Sollte man meinen. Probieren Sie ruhig ein bischen, Sie werden schon noch darauf kommen.

??????????????

Wie stellen Sie es denn an, wenn Sie einen Satz, gemeinerweise ausgerechnet aus der Sekundärdatei, nicht mehr benötigen und ihn löschen wollen?
Was geschieht in einem solchen Falle mit den Verkettungsfeldern?
Es wäre kein feiner Zug von uns, wenn wir Sie mit diesem Problem alleine ließen. Wer hätte denn auch gedacht, daß, nachdem alles so herrlich läuft, ausgerechnet das Löschen für Überraschungen sorgt.
Sie sehen, das Thema QUISAM ist lange noch nicht abgehandelt.
Es kommt noch bunter.

2.3.1.4.3 EINTRAG LöSCHEN

Das Problem ist immer dasselbe, und die Lösung gemeinhin auch:
Sollen bestimmte Daten aus einer SEQ-Datei entfernt werden, ist allgemein das Vorgehen so, daß die alten Daten ohne die zu löschenden in eine neue Datei überführt werden. Beispiel: In einer sequentiell organisierten Adressdatei mit 500 Adressen sollen 6 Einträge gelöscht werden. Da bleibt Ihnen nichts anderes übrig, als die Datei komplett Byte für Byte zu lesen, jeweils zu prüfen, ob der gerade gelesene Satz zu löschen ist und dabei alle nicht zu löschenden Sätze in eine neue Datei zu übertragen. Anders läßt es sich bei einem File dieses Typs auch kaum machen.
Anders bei einer REL-Datei.
Hier kann man einen zu löschenden Satz einfach durch Hineinschreiben eines Kennzeichens für ungültig erklären. Welches Zeichen das sein soll, liegt bei Ihnen.
So weit, so gut.
In der Regel führen aber neue Einträge zur Erweiterung der Datei, es sei denn, Sie machen sich die Mühe und durchforsten vorher alle Sätze nach einem gelöschten Eintrag, wohin Sie dann die neuen Daten schreiben können.
Dieses Verfahren spart zwar Platz, womit Sie bei der Floppy 1541 ohnehin nicht so reichlich gesegnet sind, aber es ist, abhängig von der Größe der Datei, extrem zeitaufwendig.

Wie nun haben wir das bei QUISAM gelöst?
Wie Sie aus 2.3.1.4.2 ersehen können, haben wir bereits vorgesorgt, indem wir beim Eröffnen der Anwendung den ersten freien Satz der Sekundärdatei und der Bemerkungsdatei den Variablen qs und qx zugeordnet haben.
Das erspart auf jeden Fall schon einmal die Suche. Bei der Primärdatei, die ja eine feste Länge hat, kennzeichnen wir einen freigewordenen Eintrag durch chr$(0) im Datenfeld.
Wollen wir einen neuen Eintrag vornehmen, so wird einfach das erste Zeichen des durch den Hashcode ermittelten Satzes untersucht, ob es 0 ist. Wenn ja, kommt der neue Eintrag dort hinein, wenn nicht, kommt er in die Sekundärdatei.

Soll ein Satz in Sekundär- oder Bemerkungsdatei gelöscht werden, gehen wir einfach hin und ordnen die gerade entfernte Satznummer qs, bzw. qx zu. Dadurch wird der gerade gelöschte Satz zum ersten freien Satz, der bei einem neuerlichen Eintrag in diese Datei zuerst beschrieben wird. Die alte erste freie Satznummer wird als Kettungsadresse in den nun neuen ersten freien Satz geschrieben.
Werden nacheinander mehrere Sätze gelöscht, wird der jeweils gerade gelöschte Satz vorne in die Kette eingehängt. Sie erhalten so einen Pool von freien Sätzen, die wie die Nutzdatenblöcke untereinander verkettet sind. Der zuletzt gelöschte Satz wird zum ersten freien und damit irgendwann auch zuerst wieder beschrieben, wobei alle anderen Freiblöcke eins aufrücken.
Wie Sie sehen, wird also die Datei nicht eher vergrößert, als bis sie wirklich mit gültigen Einträgen gefüllt ist.
Genauso, wie die Nutzblöcke die Satznummer des jeweils folgenden Blockes enthalten, sind auch die freien Blöcke untereinander verkettet, falls es mehr als einen gelöschten Satz in der Datei geben sollte.
Wir wollen das an Abb. 2.3-3 demonstrieren.
Dort haben wir nun aus der oben vorgestellten Adressdatei den Eintrag 'Hansen2' gelöscht. Dieser befand sich in der Sekundärdatei. Vorher war dort der erste freie Block die Nummer 6 (wie Sie in Satz 1 sehen können). Nach dem Löschen ist der erste freie Satz die Nummer 4, wo sich Hansen2 zuvor befand. Die alte Freisatznummer wird nun als Folgeblock in Satz 4 unter der Spalte 'Folge.ex' eingetragen.
Ebenso geschieht das mit der zu Hansen2 gehörenden Bemerkung im Satz 9 der Bemerkungsdatei. Die alte Freiblocknummer 12 wird in 9 als Folgeblocknummer eingetragen, während 9 in Satz 1 (und natürlich in qx) als neuer erster freier Block vermerkt wird.
Außerdem wird, wie Sie sicher bemerkt haben, auch die Folgeblocknummer bei Hansen1 in der Primärdatei geändert, da der Folgeblock nun nicht mehr Hansen2 mit der Satznummer 4 ist, sondern Hansen3 mit der Satznummer 5.

An dieser Stelle wollen wir auch erklären, was es mit der

Zahl in der Spalte 'lfd.Nr.' auf sich hat:
Wie Sie sehen, haben alle Sätze in der Primärdatei die lfd.Nr. 0. In der Sekundärdatei sind die lfd.Nr. aufsteigend bei Einträgen mit demselben Hashcode.

PRIMÄRDATEI

Satz	Folge.se	Folge.ex	lfd.Nr.	Daten
2	2	2	0	Müller1
4	0	3	0	Schultz1
15	5	8	0	Hansen1

SEKUNDÄRDATEI

Satz	Folge.se	Folge.ex	lfd.Nr.	Daten
1	4			
2	3	5	1	Meier1
3	0	6	2	Meier2
4	5	6	1	0
5	0	10	2	Hansen3

BEMERKUNGEN

Satz	Folge.ex	Daten
1	9	
2	0	Tel. 69704
3	4	hübsche Tochter
4	0	zum Essen einladen
5	0	schuldet DM 100
6	7	am 22.12. besuchen
7	0	Blumen mitnehmen
8	0	Aschenputtel
9	12	
10	11	am 20.1. Geburtstag
11	0	nicht vergessen!

Abb. 2.3-3

PRIMÄRDATEI

Satz	Folge.se	Folge.ex	lfd.Nr.	Daten
2	2	2	0	Müller1
4	0	3	0	Schultz1
14	4	9	0	Metzger1
15	5	8	0	Hansen1

SEKUNDÄRDATEI

Satz	Folge.se	Folge.ex	lfd.Nr.	Daten
1	6			
2	3	5	1	Meier1
3	0	6	2	Meier2
4	0	12	1	Metzger2
5	0	10	2	Hansen3

BEMERKUNGEN

Satz	Folge.ex	Daten
1	13	
2	0	Tel. 69704
3	4	hübsche Tochter
4	0	zum Essen einladen
5	0	schuldet DM 100
6	7	am 22.12. besuchen
7	0	Blumen mitnehmen
8	0	Aschenputtel
9	0	gute Pasteten
10	11	am 20.1. Geburtstag
11	0	nicht vergessen!
12	0	Blutwurst nicht gut

Abb. 2.3-4

Das ist äußerst brauchbar, denn so haben Sie gleich für jeden Eintrag eine eindeutige Nummer, die sich aus dem Hashcode (=Satznummer in der Primärdatei) und der lfd.Nr. zusammensetzt. So hat zum Beispiel Hansen1 die Nummer 1500 (die lfd.Nr. müssen Sie sich zweistellig denken), Hansen2 die Nummer 1501 usw.
Anwendung für diese Nummer wäre z.B. eine Artikeldatei, aus der Sie die Einträge auch nach Lagernummer suchen können. Außerdem enthebt Sie das der Mühe, die Nummern selbst vergeben zu müssen, und so können Nummern niemals doppelt auftreten, weil das QUISAM für Sie automatisch erledigt.

In Abb. 2.3-4 zeigen wir Ihnen, wie zu unserer Datei noch zwei weitere Einträge hinzukommen.
Metzger1 belegt einen neuen Platz in der Primärdatei, weil diese Satznummer bisher noch nicht aufgetaucht ist. Metzger2 jedoch nimmt in der Sekundärdatei den alten Platz von Hansen2 ein. Nun ist der erste freie Block wieder die Nummer 6.
Bei den Bemerkungen läuft es ähnlich ab. Die Bemerkung zu Metzger1 nimmt den alten Bemerkungsplatz von Hansen2 ein, während die Blutwurst von Metzger2 die Datei erweitert, da kein freigewordener Platz mehr zur Verfügung steht.

Bevor wir Ihnen die Routine zum Löschen eines Eintrages vorstellen, zunächst ein anderes kleines Unterprogramm, welches nur das Kennfeld eines spezifizierten Satzes einliest, was uns erspart, den kompletten Satz zu lesen und umständlich zu zerpflücken. Nun verstehen Sie auch, weshalb die Routine für REL lesen den Parameter rc% auswertet.
Außer den Ihnen schon bekannten Variablen kommen hier hinzu:

qt$ muß den Unterdateinamen, z.B. ".pr" enthalten.
qt enthält die Satznummer.
qy$, qz$, qy, qz dienen zum 'Durchschieben' des Unterdateinamens, bzw. der zugehörigen Satznummer. Das dient dazu, daß auf einen zuvor behandelten Satz nocheinmal zugegriffen werden kann, was später bei dem Aufbau der Satzverkettungen notwendig ist.

q1, q2, q3, q4 enthalten beim Verlassen der Routine die Inhalte der Kennfelder, wobei q4 das erste Zeichen der Nutzdaten enthält, um es später bequem auf 0 untersuchen zu können.

q1$, q2$, q3$ enthalten die Stringäquivalente von q1 bis q3, damit sie beim Zurückschreiben nicht erst wieder umgeformt werden müssen.

q5 ist eine Hilfsvariable, die hier mißbraucht wird, um die Freispeichergröße aufzunehmen (siehe auch Kommentar).

Nun das Listing, und daran anschließend der Kommentar:

```
58500 rem
58501 rem*************** vorspann lesen ****************
58502 rem
58510 qz$=qy$:qy$=qt$:qz=qy:qy=qt
58520 dn$=qn$+qt$:r=qt:rc%=6:gosub63500
58530 ifval(f1$)<>0thenreturn
58540 q1$=left$(r$,2)
58560 q1=asc(left$(q1$,1))*256+asc(right$(q1$,1))
58580 ifqt$=".ex"thenreturn
58600 q2$=mid$(r$,3,2)
58620
q2=asc(left$(q2$+chr$(0),1))*256+asc(right$(q2$,1)+chr$(0))
58640 q3$=mid$(r$,5,1):q3=asc(q3$+chr$(0))
58660 q4=asc(right$(r$,1)+chr$(0))
58680 return
```

58510 Durchschieben der Parameter. In qz$, bzw. qz stehen immer die Parameter vom vorigen Durchlauf.

58520 Aufbau der Parameter für REL lesen. Da die Kennfelder zusammen höchstens 6 Zeichen umfassen, werden auch nur 6 Bytes gelesen (rc%=6).

58530 Falls der Satz nicht gelesen werden konnte, weil er z.B. nicht vorhanden war, kann die Routine hier abgebrochen werden.

58540-

58660 Hier werden die gelesenen Zeichen zerlegt und den Variablen q1 bis q4, bzw. q1$ bis q3$ zugeordnet.

Die nächste Routine nun wäre das Löschen. Dazu noch ein paar Worte:
Da in unserem Dateisystem die Hashcodes durchaus doppeldeutig sein können, ist die Satznummer allein als Zugriffskriterium ungeeignet. Es könnte leicht den Falschen erwischen. Deshalb wird grundsätzlich nur nach der vollständigen Nummer des Eintrages gelöscht, die ja eindeutig ist, wie oben ausgeführt.
Das zwingt Sie dazu, den Eintrag zuerst nach Schlüsselbegriff zu suchen (wie, das folgt später), um so die Nummer zu erfahren, und dann zu löschen.
Als Variable tritt also hier, außer den schon bekannten, nur noch

qn auf, die Sie mit der Nummer versehen müssen.
Nun das Listing:

```
55000 rem
55001 rem************* eintrag loeschen *********
55020 qn%=val(right$(str$(qn),2))
55040 qt=(qn-qn%)/100
55060 qt$=".pr":qn=1:gosub58500
55080 ifqn%<>0thenqt=q1:qn=0:goto55300
55100 ifq4=0thenreturn
55120 r$=q1$+chr$(0)+chr$(0)+q3$+chr$(0)
55140 gosub63000:ifq1=0andqt$=".pr"then55170
55150 dn$=qn$+qz$:r=qz:rc%=255:gosub63500
55160 r$=q1$+mid$(r$,3,ql%+3):gosub63000
55170 onqngoto55400
55180 rh%=qs/256:rl%=qs-256*rh%:dn$=qn$+qy$:r=qy
55190 r$=q1$+chr$(rh%)+chr$(rl%)+q3$+chr$(0)
55200 gosub63000:qs=qt
55220 rh%=qs/256:rl%=qs-256*rh%
55240 r=1:r$=chr$(rh%)+chr$(rl%)
55260 gosub63000:goto55400
```

```
55300 qt$=".se":gosub58500
55320 ifqn%=q3then55100
55340 qt=q1:goto55300
55400 qt=q2:ifqt=0thenreturn
55420 qt$=".ex"
55440 gosub58500
55460 ifq1<>0thenqt=q1:goto55440
55480 rh%=qx/256:rl%=qx-256*rh%
55500 r$=chr$(rh%)+chr$(rl%)
55520 gosub63000:qx=q2
55540 r=1:r$=q2$:gosub63000
55560 return
```

55020-
55040 Aus der vollständigen Nummer qn wird die lfd.Nr. qn% (letzte beide Stellen von qn) und die Satznummer qt der Primärdatei gebildet.

55060 Aus der Primärdatei wird der Kennsatz gelesen.

55080 Bei qn%<>0 befindet sich der Satz in dr Sekundärdatei. Dortgeht es dann witer.

55100 Ist der Satz bereits frei (q4=0), braucht nichts weiter unternommen werden.

55120-
55140 Der Satz wird gelöscht, wobei jedoch die Kettungsadressen erhalten bleiben müssen, da noch Folgeblöcke existieren können. Falls es sich um die Primärdatei handelte, braucht keine Rückwärtskettung vorgenommen werden (qt$=".pr").

55150-
55160 Der einen Durchgang früher gelesene Satz wird wieder hervorgeholt und mit dem Kettungsfeld des zu löschenden Satzes versehen, damit die Folge durch das Löschen nicht unterbrochen wird. (siehe auch Abb. 2.3-3)

55180-
55260 Die Nummer des bisherigen ersten freien Satzes wird in den gelöschten Satz eingetragen und dessen Satznummer bildet den neuen ersten Freiblock und wird in Satz 1,

bzw. qs hinterlegt.

55300-
55340 Hier wird die Sekundärdatei anhand der Verkettungen solange durchsucht, bis der gesuchte Satz gefunden ist (q%=q3).

55400 Gibt es keine Bemerkungen (q2=0), ist nichts weiter zu tun, andernfalls geht es dort weiter.

55420-
55560 Die Bemerkungen werden nicht physikalisch gelöscht. Es wird lediglich das erste Glied der Kette zum ersten freien Satz, und das letzte Glied bekommt die alte Freiblocknummer eingetragen. Satz 1 und qx werden entsprechend versorgt.

2.3.1.4.4 SATZ EINTRAGEN

Wir haben die Erläuterung des Löschens deshalb vor das Anlegen von Einträgen gezogen, weil das wesentlich zum Verständnis des Schreibmechanismus ist.
Der Eintrag eines Satzes hätte zwar thematisch vor den vorigen Abschnitt gehört, jedoch sind wir der Meinung, daß es logisch hier besser aufgehoben ist, Wir hoffen hierfür auf Ihr Verständnis.

Es folgt das Eintragen eines Stammsatzes.
Einziger Übergabeparameter ist
qd$, worin der zu Schreibende Satz enthalten sein muß. Die übrigen Variablen sind bekannt.

Zunächst wieder das Listing, daran anschließend der Kommentar.

```
57000 rem
57001 rem**************** eintrag stamm ***************
57002 rem
57020 gosub58800:qn%=0
57040 qt$=".pr":qt=hc:gosub58500
57060 ifq4<>0thenqt$=".se":goto57260
57080 rc%=255:gosub63500
57100 r$=left$(r$,5)+left$(qd$,ql%):gosub63000
57200 return
57260 ifq3=14thenqn%=13
57280 ifq3>qn%+1thenq6=qt:dn$=qn$+qz$:r=qz:goto57330
57310 qn%=q3:ifq1<>0thenqt=q1:gosub58500:goto57260
57320 q6=0
57330 rc%=255:gosub63500
57340 rh%=qs/256:rl%=qs-256*rh%
57360 r$=chr$(rh%)+chr$(rl%)+mid$(r$,3,ql%+3):gosub63000
57400 qt=qs:gosub58500
57410 qn%=qn%+1:ifqn%=13thenqn%=qn%+1
57420 ifval(f1$)<>0then57440
57430 ifq2<>0thenqs=q2:goto57480
57440 qs=qs+1:rh%=qs/256:rl%=qs-256*rh%
57450 ifrh%=13thenqs=qs+256
57460 ifrl%=13thenqs=qs+1
57480 rh%=q6/256:rl%=q6-256*rh%
57500 r$=chr$(rh%)+chr$(rl%)+chr$(0)+chr$(0)
57510 r$=r$+chr$(qn%)+left$(qd$,ql%)
57520 gosub63000
57540 rh%=qs/256:rl%=qs-256*rh%
57560 r$=chr$(rh%)+chr$(rl%):r=1:gosub63000
57580 return
```

57020-
57200 Nachdem aus dem Satzinhalt die Satznummer errechnet wurde, wird, falls der entsprechende Platz in der Primärdatei frei ist (q4=0), der neue Satz dort ingetragen, andernfalls wird die Sekundärdatei bemüht.

57260-
57310 Hier wird die Sekundärdatei anhand der Kettung

durchgehangelt, um die nächste freie lfd.Nr. zu finden.

57330-
57360 Der Satz mit der höchsten gefundenen lfd.Nr. wird mit qs als Folgesatznummer versehen, da dort ja der neue Satz eingetragen wird.

57400-
57520 Der neue Satz wird eingetragen. Abhängig davon, ob dieser Platz einem gelöschten Satz gehörte (f1$="00") oder ein neuer vom Dateiende ist, wird qs entweder erhöht oder mit der Kettungsnummer q2 geladen.

57540-
57580 Satz 1 wird auf den aktuellen Stand gebracht.

Nun häufen sich die Programme ein wenig, aber bald ist es geschafft.
Was beim Eintrag noch fehlt, sind die Bemerkungen.
Diese Routine braucht nur mit dem Text in qx$ angesprungen werden.
Die sonst noch benötigten Parameter werden dem zuvor bearbeiteten Stammsatz entnommen. Sollen also Bemerkungen zu einem Stammsatz erst zu einem späteren Zeitpunkt nachgetragen werden, so muß dieser Stammsatz zuvor gelesen werden, damit die Parameter qt und qt$ richtig versorgt sind. Es geht los:

```
56000 rem
56001 rem*************** eintrag anhang ***********
56002 rem
56010 qf%=0:ifqt$=".ex"thenqf%=1:return
56020 gosub58500:q3=0
56040 qt$=".ex":ifq2<>0then56200
56060 rc%=255:gosub63500
56080 rh%=qx/256:rl%=qx-256*rh%
56100 r$=left$(r$,2)+chr$(rh%)+chr$(rl%)+mid$(r$,5,ql%+1)
56120 gosub63000:q3=1:goto56260
56140 ifq2=0thenq2=qt:goto56260
```

```
56200 q1=q2:qt=q2:gosub58500
56220 q2=q1:goto56140
56260 qt=qx:gosub58500
56280 q4=qx:ifval(f1$)<>0then56300
56290 ifq1<>0thenqx=q1:goto56340
56300 qx=qx+1:rh%=qx/256:rl%=qx-256*rh%
56310 ifrh%=13thenqx=qx+256
56320 ifrl%=13thenqx=qx+1
56340 r$=chr$(0)+chr$(0)+left$(qx$,qx%):gosub63000
56360 onq3goto56500
56380 r=q2:rc%=255:gosub63500
56400 rh%=q4/256:rl%=q4-256*rh%
56420 r$=chr$(rh%)+chr$(rl%)+mid$(r$,3,qx%)
56440 gosub63000
56500 rh%=qx/256:rl%=qx-256*rh%
56540 r=1:r$=chr$(rh%)+chr$(rl%):gosub63000
56560 return
56600 q2=qt:q3=0:qf%=0
56620 ifqt$<>".ex"thenqf%=1:return
56640 goto56260
```

56020-
56120 Wenn der Stammsatz noch über keine Bemerkungen verfügt (q2=0), wird als Kettungsadresse qx dort eingetragen, da ja dort auf jeden Fall die Bemerkung hinkommt.

56140-
56220 Die schon für diesen Stamm vorhandenen Bemerkungen werden bis zur derzeit letzten durchsucht, um die Verbindung zur neuen herstellen zu können.

56260-
56340 Der neue Eintrag wird vorgenommen und qx abhängig von f1$ (Datei um neuen Satz erweitert?) versorgt.

56360-
56440 Existierten bereits Bemerkungssätze (Merker q3=0), muß deren letzter mit der neuen Satznummer versehen werden.

56500-
56560 Satz 1 wird mit qx versorgt.

56600 Das ist der Einsprung in die Routine, wenn für den gleichen Stammsatz weitere Bemerkungen erfolgen sollen. Die Kettungsparameter sind noch vorhanden, sodaß wir uns nur auf den Neueintrag beschränken brauchen. Es muß auf jeden Fall unmittelbar vorher ein Bemerkungseintrag über 56000 erfolgt sein.

Was zur Vollständigen Funktion des Paketes noch fehlt, sind lediglich die Suchroutinen.

2.3.1.4.5 SATZ SUCHEN

Die Suche, sowohl nach einem Stammeintrag als auch nach Bemerkungen, kann nach zwei Kriterien erfolgen, nämlich nach der vollständigen Nummer und nach dem Schlüssel.
Das Wiederauffinden von Bemerkungen führt, bis auf die Ausnahme des fortgesetzten Lesens, immer über die Stammsatzsuche, da ja dort der Verweis auf die erste Bemerkung zu finden ist.
Beginnen wir mit der Stammsatzsuche nach Schlüssel.
Das Suchkriterium, z.B. der Nachname, wird in qk$ übergeben, den Rest rledigt die Routine, die nicht nur den vollständigen Satz in qd$ zurückliefert, sondern auch in qn die Nummer des Eintrages, der dann künftig als Suchkriterium oder als Löschparameter vrwendet werden kann.
qf% ist ein Flag, welches bei <>0 die erfolglose Suche signalisiert.

```
54000 rem
54001 rem***** stammeintrag nach schluessel suchen ***
54002 rem
54020 hc$=left$(qk$+"     ",5)
54040 gosub58840
54060 qt$=".pr":qt=hc:qf%=0
54080 gosub58500
54100 rc%=255:gosub63500
54120 ifmid$(r$,qk%+5,len(qk$))=qk$then54200
```

```
54140 ifq1=0thenqf%=1:return
54160 qt=q1:qt$=".se":goto54080
54200 qn=hc*100+q3
54220 qd$=mid$(r$,6,q1%)
54240 return
```

54020-
54060 Die Suche wird in der Primärdatei begonnen, nachdem die Satznummer aus qk$ ermittelt wurde.

54100-
54160 Die Sätze werden nun nacheinander gelesen, beim ersten mal aus der Primärdatei, ab dem zweiten mal aus der Sekundärdatei, wobei der Teil des Satzes, der gemäß qk% als Schlüsselkriterium dient, mit qk$ verglichen wird.

54200-
54240 Aus der lfd.Nr. wird zusammen mit hc die komplette Nummer gebildet, die Nutzdaten aus r$ herausgezogen und in qd$ abgeliefert.

Die Suche eines Stammsatzes nach Nummer sieht ähnlich aus.
Hier wird jedoch die zu suchende Nummer in qn übergeben.
Als Kommentar mag genügen, daß etwa in der gleichen Weise wie oben vorgegangen wird, jedoch wird nur jeweils das Kennfeld gelesen und auf Übereinstimmung mit der lfd.Nr. (diese wurde anfangs von qn abgetrennt) untersucht (qn%=q3). Anschließend wird auch hier der vollständige Satz in qd$ übergeben.

```
54500 rem
54501 rem******** stammeintrag nach nummer suchen *****
54502 rem
54520 qn%=val(right$(str$(qn),2))
54540 qt=(qn-qn%)/100:qf%=0
54560 qt$=".pr":gosub58500
54580 ifqn%=0then54700
54590 ifq1=0thenqf%=1:return
```

```
54600 qt$=".se"
54620 qt=q1:gosub58500
54640 ifq3=qn%then54700
54660 ifq1=0thenqf%=1:return
54680 goto54620
54700 ifq4=0thenqf%=1:return
54710 rc%=255:gosub63500
54720 goto54220
```

Die Suche nach den Bemerkungen anhand der unterschiedlichen Kriterien haben wir zusammengefaßt, da deren Hauptbestandteil die beiden oben vorgeführten Routinen sind. Nachdem Übereinstimmung festgestellt wurde, wird der anhand der Kettungsadresse (q2) aufgefundene erste Bemerkungssatz übergeben.

Folgeeinträge werden über 53200 aufgesucht, da ja alle Parameter noch vorhanden sind. Das Ende wird wieder über qf%<>0 angezeigt.

```
53000 rem
53001 rem******** anhang nach schluessel suchen *****
53002 rem
53020 qf%=0:gosub54000
53040 goto53200
53100 rem
53101 rem******* anhang nach nummer suchen *********
53102 rem
53120 qf%=0:gosub54500
53200 rem
53201 rem********* folgeeintrag suchen ************
53202 rem
53220 ifqf%thenreturn
53240 ifq2=0thenqf%=2:return
53260 r=q2:dn$=qn$+".ex":rc%=255
53280 gosub63500
53300 qx$=mid$(r$,3,qx%)
53320 q2=asc(left$(r$,1))*256+asc(mid$(r$,2,1))
```

53340 return

2.3.1.4.6 WOFÜR EIGNET SICH QUISAM?

Die Eignung von QUISAM für bestimmte Anwendungen ist, wie Sie den Ausführungen entnehmen können, wesentlich von der Dimensionierung der Primärdatei abhängig.
Für beispielsweise ein Kassenjournal ist es nur notwendig, als Suchkriterium das Tagesdatum herzunehmen. Die laufenden Einnahmen und Ausgaben schreiben Sie in die Bemerkungsdatei. Damit erreichen Sie sowohl beim Schreiben als auch beim Lesen die maximale Geschwindigkeit, da die Datei geöffnet bleibt, wodurch die Zugriffe schneller ablaufen, da das öffnen und Schließen einer Datei sehr zeitaufwendig ist (erinnern Sie sich: die VC-1541 kann nur in REL-File zu einer Zeit verwalten). Da der Schlüssel selten wechselt, darf die Primärdatei ruhig klein bleiben, da ein Ausweichen auf die Sekundärdatei in Bezug auf die gesamte Verarbeitungsdauer nur wenig Zeit beansprucht.
Anders sieht es bei der Adressverwaltung aus, bei der die Schlüssel häufig wechseln. Hier ist eine möglichst große Primärdatei nützlich, damit diese geöffnet bleiben kann. Ein Wechsel zur Sekundärdatei erfordert nämlich wieder CLOSE und OPEN.

Nun erhebt sich bei Ihnen sicher die Frage, wie Sie QUISAM Ihren eigenen Zwecken nutzbar machen können, und ob Sie bei allem Komfort überhaupt noch Platz für Ihre eigenen Programme haben.
Die Frage nach dem Platz ist schnell geantwortet: Alle im gesamten Kapitel 2 vorgestellten Routinen zusammen umfassen nur ca. 15k. Sicher, das erscheint auf den ersten Blick viel, aber dafür kann Ihre spezielle Anwendung minimal ausfallen. Ein prägnantes Beispiel dafür finden Sie am Ende des Buches im Programmteil unter 'Literaturverzeichnis'. Sie können dort sehen, wie mit nur wenigen zusätzlichen Zeilen eine komplette Quellenverwaltung mit Bildschirmmaske und allem Drum und Dran unter Verwendung von QUISAM realisiert

wurde.
Prinzipiell gehen Sie so vor, daß Sie alle Routinen laden (dazu sollten sie alle miteinander einen Komplex bilden) und dann, wie gewohnt, Ihr Programm hinzuschreiben. Komfortabler geht es natürlich mit EXBASIC o.ä., denn damit können Sie mit der Funktion MERGE nur die wirklich benötigten Routinen Ihrem Programm hinzufügen.
Es gibt nur relativ wenige Einsprünge, und für einen zusammenhängenden Themenkomplex sind die Übergabevariablen immer dieselben. Nach kurzer Zeit werden Sie virtuos damit jonglieren können. Überflüssig, zu bemerken, daß Sie natürlich nicht die in den Programmen benutzten Variablen anderweitig verändern dürfen.

Damit endet dieses Kapitel.
Wir hoffen, daß es nicht allzu trocken war, aber wir denken, daß sich die Mühe für Sie und für uns gelohnt hat.

WICHTIGER HINWEIS: QUISAM ist eine vom Leiter der DATA BECKER Entwicklungsabteilung, Klaus Gerits, eigens für dieses Buch geschaffene Neuentwicklung. Trotz ausführlicher Tests können natürlich grundsätzlich noch Fehler enthalten sein. Für Anregungen und konstruktive Kritik, bitte nur in schriftlicher Form, ist der Autor stets dankbar.

2.3.2 DIE AUSGABE AUF DRUCKER

Die allermeisten Programme enden damit, daß sie, nachdem Daten eingegeben, vom Programm mißhandelt und auf Floppy hin- und hergeschaufelt wurden, diese Daten schließlich zu Papier bringen, etwa nach dem Motto 'Was du schwarz auf weiß besitzt ....'.
Jedoch ist für den Programmierer kein Thema so banal und gleichzeitig so tückisch wie die Ausgabe auf Drucker.
Solange Sie nur für den Eigenbedarf programmieren, ist alles klar, denn Sie kennen Ihren eigenen Drucker genau.
Problematisch wird es erst, wenn Sie nicht wissen, auf welchem Drucker die Ausgabe erfolgen soll, denn diese Geräte unterscheiden sich in der Programmierung gewaltig.

2.3.2.1 DRUCKER GLEICH DRUCKER?

Kennen Sie dieses Kribbeln im Rücken, das allmählich bis in die Haarspitzen steigt und beim Passieren des Kopfes auf dem Gesicht eine nicht zu übersehende Rötung hinterläßt, die uns bei Kindern oft auf ein schlechtes Gewissen tippen läßt?
Sehen Sie diese gesunde Hautfarbe einmal bei einem Softwareanbieter auf die Frage hin, ob das Programm X auch mit dem Drucker Y zusammenarbeite, liegen Sie mit dem Tip auch hier nicht ganz falsch.
Was ist die Ursache?
Bei Mixed Hardware (mehrere Geräte unterschiedlicher Hersteller in einem System) kann man von vornherein unterstellen, daß ein z.B. Centronics-Drucker, abgesehen von dem physikalischen Anschluß, nicht unbedingt zu Ihrem C64 paßt. Das fängt schon bei den Graphikzeichen an. Davon soll hier auch nicht die Rede sein.
Viel schwerer wiegt, daß auch nicht alle zum Anschluß an den C64 vorgesehenen Commodore-Drucker programmtechnisch gleich zu handhaben sind. Um ein konkretes Beispiel zu nennen:
Sie haben ein Programm mit Druckausgabe für den im VC-Bereich weit verbreiteten Drucker SEIKO GP-100VC erstellt. Da Sie Groß- Kleinschrift wünschen und es Ihnen zu

lästig war, vor jede Druckzeile chr$(17) zu setzen, was bedeutet, daß eben diese Zeile in der erwähnten Schriftart gedruckt werden soll, haben Sie den Drucker einfach mit OPEN1,4,7 eröffnet und sofort lustig Daten ausgegeben, was für diesen Drucker heißt, daß die über die Sekundäradresse 7 geleiteten Daten samt und sonders in der Groß-Kleindarstellung gedruckt werden.

Das gleiche Programm mit Ausgabe auf einen VC-1526 führt zu einer von wüsten Beschimpfungen begleiteten Fehlersuche: Der Drucker rührt sich nicht. Die Ursache ist einfach die, daß der VC-1526 über die Sekundäradresse 7 keine Daten annimmt, da diese nur als Umschaltadresse für die Betriebsart gedacht ist. Um das zu erreichen, was beabsichtigt war, ist die Sequenz

```
10open1,4,7:print1:close1:rem Umschalten auf Groß-Klein
20open1,4
.
30print#1,"daten"
```

vonnöten. Andersherum hat ein stolzer Besitzer des VC-1526 dessen exzellente Fähigkeit zur Formatierten Ausgabe über Sekundäradresse 1 und 2 genutzt. Diese Adressen wiederum sind für den GP-100VC keineswegs Anlaß, das gewünschte Ergebnis zu produzieren.

Im Falle Groß- Kleinschreibung ist es also nie falsch, die Betriebsart über chr$(17) zu steuern, weil das die meisten Drucker beherrschen. Anders ist es mit einer Feldbündigen Ausgabe von Daten. Das können die allerwenigsten, im Bereich VC bis jetzt sogar nur der VC-1526.

Da diese Möglichkeit den Komfort beträchtlich erhöht und das Erscheinungsbild eines bedruckten Blattes erheblich verbessert, insbesondere bei Formularen jeder Art, wollen wir uns ein wenig näher damit befassen, und überlegen, wie diese Möglichkeit, wenn schon nicht im Drucker vorhanden, wenigstens programmtechnisch zu simulieren. Genau das soll Gegenstand der folgenden Abschnitte sein:

Wie stellen Sie es an, daß das Druckbild vernünftig ist, ohne daß Sie auf einen intelligenten Drucker zurückgreifen müssen, der bedarfsweise die Textblöcke auch ausrichtet.

Was jeder Drucker kann, ist, die Zeichen so zu

wiederzugeben, wie sie kommen. Es ist also Ihre Aufgabe, die Daten so aufzubereiten, daß das Erscheinungsbild dem zugedachten Zweck entspricht.
Die Ausgabe eines Briefes ist, vom Randausgleich einmal abgesehen, die unproblematischste Form der Ausgabe.
Anders sieht es bereits beim Drucken von Rechnungen aus. Da müssen Textteil und Zahlenteil ausgerichtet werden, nämlich in der Regel der Text linksbündig, die Zahlen rechtsbündig.
Welche Möglichkeiten der Aufbereitung machbar sind, soll nun diskutiert werden.

2.3.2.2 TABULATOR

Eine Möglichkeit, das Erscheinungsbild zu beeinflussen, bietet Ihnen schon Ihr Rechner von Haus aus: Die Funktion TAB.
Leider verhält sich diese bei Ausgabe auf den Bildschirm anders als beim Drucker.
Möglicherweise ist es Ihnen schon aufgefallen, daß beispielsweise die Zeile

```
print"hallo"tab(10)"wie geht es"
```

auf dem Bildschirm ganz ordentlich

```
hallo     wie geht es
```

produziert, auf dem Drucker jedoch

```
hallo          wie geht es
```

Wie ist das möglich? Ganz einfach, das Betriebssystem wandelt den tab(x) bei der Ausgabe auf ein anderes Gerät als den Bildschirm in ein spc(x) um.
Tab bedeutet ja eine absolute Position vom linken Rand, während spc eine Distanz vom augenblicklichen Stand des Cursor, bzw. Druckkopfes bestimmt.
Abhängig von der Länge des ersten Textes wird sich auch der

Anfang des zweiten Textes verschieben.
Was also ist zu tun?
Eine Möglichkeit gibt es, die jedoch nur auf Commodore-Drucker anwendbar ist:
die Bestimmung der absoluten Position vom linken Rand mittels Wagenrücklauf ohne Zeilenvorschub und anschließendes Drucken des zweiten Zeilenteiles.
Einen Rücklauf des Druckkopfes ohne Papiertransport können Sie durch Senden von chr$(141) erreichen. Am konkreten Beispiel sieht das für den Druck einer Rechnungszeile so aus:

```
10 open1,4
20 print#1,"Verlängerungsschnur"chr$(141)spc(40)"DM 6.45"
```

Der Text wird nun linksbündig gedruckt, und die Preisangabe ab der Spalte 40.
Das verdanken wir dem einfachen Trick, daß wir erst den Text gedruckt, dann einen Wagenrücklauf ohne Zeilenvorschub verursacht, anschließend ein Vorrücken um 20 Zeichen befohlen und dann den Text gedruckt haben.
Sie sehen also, daß Sie bei der Ausgabe auf Drucker die Wirkung eines echten TAB durch Voranstellen eines chr$(141) erreichen können.
Sie haben aber damit erst erreicht, daß die gewünschten Texte an den durch spc(x) festgelegten Stellen beginnen, was jedoch nicht in allen Fällen auch zweckmäßig ist.
Was wir damit meinen, sehen Sie im nächsten Abschnitt.

2.3.2.3 DEZIMALPUNKTE

Um bei der Rechnungszeile aus dem vorigen Abschnitt zu bleiben:
Angenommen, eine nach dem gleichen Schema übergebene Zeile enthielte als Betrag den String "DM 66.45". Der Betrag würde zwar wieder an der gleichen Position beginnen, jedoch würde der Dezimalpunkt gegenüber der darüberliegenden Zeile um eine Stelle nach rechts rücken. Stellen Sie sich vor, so würde eine Rechnung mit Beträgen unterschiedlicher Vorkommastellenzahl gedruckt.

Das Ergebnis wäre als Exponat einer Kuriositätensammlung geeignet, nicht jedoch zur Dokumentation Ihrer finanziellen Forderungen.
Wie fast alles, ist natürlich auch dieses Problem jeweils programmtechnisch zu lösen, indem Sie zunächst aus der Zahl einen String machen, diesen dann auf die Lage des Dezimalpunktes untersuchen und den spc-Parameter davon abhängig machen, wobei natürlich die unterschiedlichen Positionen einer Zeile auch unterschiedlich behandelt werden müssen.
Diesen nicht unerheblichen Aufwand haben wir Ihnen wieder abgenommen, und zwar auf die für Sie allerbequemsten Weise, wovon Sie sich im nächsten Abschnitt überzeugen können.

2.3.2.4 FORMATIERUNG

Unter einem Format, hier Druckformat, verstehen wir das Erscheinungsbild einer ganzen Zeile.
Bei der Rechnung kann eine Zeile etliche Felder haben. Das beginnt mit der Positionsnummer, dann kommt die Menge, dann die Artikelbezeichnung, dann der Einzelpreis und abschließend der Gesamtpreis.
Wenn Sie das nach obigem Schema realisieren wollten, hätten Sie gut zu tun.
Wie einfach wäre es, wenn man das Format einer Zeile unterschieden nach alphanumerischen und rein numerischen Feldern einmalig aufbauen könnte und die Ausgabe unter allen Umständen diesem Format entspräche.
Das folgende Unterprogramm nimmt Ihnen diese Arbeit ab, und der resultierende Ausdruck kann sich sehen lassen.

```
48000 rem
48001 rem******************* formatieren *************
48002 rem
48020 iffz%=0thenfz%=1:fl%=len(fo$):fp$=""
48040 fi%=len(fi$):fb$="":fc%=1
48050 ifmid$(fo$,fz%,1)=" "thenfz%=fz%+1:
      fb$=fb$+" ":goto48050
48060 fm$=mid$(fo$,fz%,1)
48080 fv%=0:fh%=0:fa%=0:fb%=0:ff%=0
48100 ifmid$(fo$,fz%,1)="."thenfz%=fz%+1:ff%=2:goto48180
48120 iffz%>fl%thenfa%=fa%+1:fz%=fz%+1:goto48240
48140 ifmid$(fo$,fz%,1)=" "then48240
48160 fa%=fa%+1:fz%=fz%+1:goto48100
48180 ifmid$(fo$,fz%,1)=" "orfz%>fl%then48240
48200 fb%=fb%+1:fz%=fz%+1
48220 goto48180
48240 ifmid$(fi$,fc%,1)="."thenfc%=fc%+1:ff%=ff%+4:goto48320
48260 fv%=fv%+1:fc%=fc%+1
48280 iffc%>fi%thengoto48380
48300 goto 48240
48320 iffc%>fi%then48380
48340 fh%=fh%+1:fc%=fc%+1
48360 goto48320
48380 iffm$="a"then48780
48400 l2=fa%-fv%:ifl2<0thenff%=ff%+1:goto48720
48420 ifl2=0then48460
48440 forl1=1tol2:fb$=fb$+" ":nextl1
48460 iffv%=0then48520
48480 l2=fv%:forl1=1tol2
48500 fb$=fb$+mid$(fi$,l1,1):nextl1
48520 ifff%=0then48920
48540 ff%=0:fb$=fb$+".":l2=fv%+2
48560 iffb%=0then48920
48580 iffb%=fh%thenl3=fb%:goto48620
48600 l3=-fh%*(fb%>fh%)-fb%*(fh%>fb%)-1
48620 forl1=l2tol2+l3
48640 fb$=fb$+mid$(fi$,l1,1):nextl1
48660 iffh%>=fb%then48910
48680 l2=fb%-fh%:forl1=1tol2
```

```
48700 fb$=fb$+"0":nextl1:goto48910
48720 l2=fa%+fb%:if(ff%and2)thenff%=1:l2=l2+1
48740 forl1=1tol2:fb$=fb$+"*"
48760 nextl1:goto48910
48780 if(ff%and4)thenfv%=fv%+fh%+1
48800 l2=fv%:forl1=1tol2
48820 ifmid$(fi$,l1,1)<>" "thenl3=l1:l1=l2
48840 nextl1
48860 l2=-fv%*(fa%>fv%)-fa%*(fv%>fa%)
48870 ifl2=0thenl2=fv%
48880 forl1=l3tol3+l2
48900 fb$=fb$+mid$(fi$,l1,1):nextl1
48902 iffv%>=fa%then48910
48904 l3=fa%-l2:forl1=1tol3
48906 fb$=fb$+" ":nextl1
48910 iffz%>fl%thenfz%=0:goto48960
48920 l2=fz%:l3=fl%:forl1=l2tol3
48930 ifmid$(fo$,l1,1)<>" "thenfz%=l1:l1=l3:goto48950
48940 fb$=fb$+" "
48950 nextl1
48960 fp$=fp$+fb$:ff%=(ff%and1)
48970 return
```

Da das Programm viele interne Variablen enthält, wollen wir Ihnen nur diejenigen vorstellen, die den Ablauf der Routine wesentlich beeinflussen.

fo$ muß das von Ihnen gewünschte Zeilenformat enthalten.

fi$ müssen Sie mit den Daten versorgen, die Sie gerne ausgerichtet hätten.

fb$ enthält den gerade aufbereiteten Teilstring, während

fp$ den vollständigen Druckstring enthält.

fz% ist der Zeiger auf den gerade bearbeiteten Terminus in fo$.

ff% signalisiert einen Überlauf der Vorkommastellen (ff%<>0).

fa% enthält die Anzahl der Vorkommastellen des gerade bearbeiteten Feldes in fo$.

fb% wie oben, jedoch Nachkommastellen.
fv% Anzahl der Vorkommastellen in fi$.
fh% Anzahl der Nachkommastellen in fi$.

Der Kommentar:

48020-
48080 Abhängig vom erstmaligen Einsprung (fz%=0) werden einige Variable rückgesetzt.

48100-
48160 Die Vorkommastellen des gerade bearbeiteten Feldes von fo$ nach fa%.

48180-
48220 Wie oben, jedoch Nachkommastellen nach fb%.

48240-
48300 Anzahl der Vorkommastellen in fi$ nach fv%.

48320-
48360 Anzahl der Nachkommastellen in fi$ nach fh%.

48400-
48440 Hier beginnt die Ausrichtung numerischer Felder. Falls fi$ weniger Vorkommastellen als fo$ enthält, wird mit Leerzeichen aufgefüllt.

48460-
48500 Die gültigen Vorkommastellen kommen nach fb$.

48560-
48640 Die gültigen Nachkommastellen kommen nach fb$.

48660-
48700 Falls fi$ weniger Nachkommastellen als fo$ enthält, wird mit Nullen aufgefüllt.

48720-
48760 Hatte fi$ mehr Vorkommastellen als fo$, wird das gesamte Feld mit '*' gefüllt, damit es auf dem Ausdruck sofort ins Auge fällt. Sie haben in diesem Falle das Feld zu knapp bemessen. Dem Programm wird dieser Zustand durch ff%<>0 signalisiert.

48780-
48840 Hier beginnt die Aufbereitung für Alphafelder. Zunächst werden führende Leerstellen aus fi$ unterdrückt, damit die sichtbaren Zeichen auch

tatsächlich am Feldanfang beginnen.

48860-
48900 Aus fi$ werden Zeichen gemäß fo$ nach fb$ übertragen. Übrige Zeichen aus fi$ werden abgeschnitten, übrige Zeichen aus fo$ mit Leerstellen aufgefüllt.

48910-
48950 Gemäß der Distanz zum nächsten Feld in fo$ wird fb$ mit Leerstellen aufgefüllt.

48960-
48970 fb$ wird auf fp$ addiert.

Die Bedienung gestaltet sich recht einfach:
Stellen Sie fest, wieviele Felder es in Ihrer Zeile gibt, und ob diese Alphadaten oder Zahlen enthalten sollen.
Anschließend halten Sie dieses Format in fo$ fest. Das sähe für die oben beschriebene Rechnung so aus:

```
fo$="99  999  aaaaaaaaa  999.99  9999.99"
```

Sie sehen schon, daß numerische Felder mit '9' gekennzeichnet werden, Alphafeldermit 'a'. Die unterschiedliche Behandlung besteht darin, daß die Daten in Alphafeldern linksbündig wiedergegeben werden, numerische jedoch rechtsbündig in Übereinstimmung des Dezimalpunktes mit seiner Lage in fo$. Nun brauchen Sie nur noch fi$ mit Ihren Daten zu versorgen und das Programm sooft anzuspringen, wie Sie Felder in fo$ definiert haben, natürlich mit immer anderen Daten in fi$.
Am Schluß der Prozedur haben Sie in fp$ die vollständige Druckzeile.
Das könnte in Ihrem Programm dann so aussehen:

```
10open1,4
20fo$="99  999  aaaaaaaaa  999.99  9999.99"
30fi$=str$(po):gosub48000:rem positionsnummer
40fi$=str$(an):gosub48000:rem anzahl
50fi$=ar$:gosub48000:rem artikelbezeichnung
60fi$=str$(ep):gosub48000:rem einzelpreis
70fi$=str$(an*ep):gosub48000:rem ges.-preis
80print#1,fp$
```

Sie sehen, daß die Routine fünfmal angesprungen wird, nämlich sooft, wie Felder in fo$ definiert sind.
Die Definition in fo$ brauchen Sie natürlich nur einmal übergeben. Sie bleibt solange erhalten, bis Sie ein anderes Format in fo$ hinterlegen.

Eine große Hilfe bietet diese Routine auch beim Etikettendruck. Gewöhnlich nimmt man zum Drucken von Selbstklebeetiketten nur solche, wo nur je ein Etikett in der Reihe auf dem Träger montiert ist. Diesen spannt man dann linksbündig in den Drucker ein und beschreibt die Etiketten mit ganz normalen Print-Anweisungen. Die erste Hilfe der Formatierung besteht darin, das Format so zu wählen, daß bei längeren Sätzen ein Druck über das Etikett hinaus verhindert wird. Da es sich aber dennoch um recht kurze Zeilen handelt, die jeweils untereinander gedruckt werden, wird das mögliche Zeilenformat des Druckers schlecht genutzt, d.h. der Druckkopf muß für jede Zeile je eines Etiketts einmal bewegt werden, und anschließend wird ein Zeilentransport ausgeführt. Das dauert alles seine Zeit.
Erheblich schneller geht es, wenn Sie, der maximalen Zeilenlänge des Druckers entsprechend, einen Träger besorgen, auf dem Etiketten zu zweit oder zu dritt nebeneinander montiert sind (durchaus handelsüblich). Nun wird die Druckaufbereitung für den Programmierer äußerst kompliziert, denn abgesehen davon, daß nun die Daten einer Zeile für je zwei (oder drei) Etiketten bereitgestellt werden müssen, ist es unumgänglich, die Druckzeile vorher zusammenzusetzen in der Form, daß die Zeilenanfänge jedes Etiketts auch untereinanderstehen. Drei Verfahren sind dazu möglich:
Falls der Drucker selbst eine Tabelliereirichtung besitzt, können Sie dessen Tabulatorpositionen vorwählen und dann munter drucken. Leider besitzen die Commodore-Drucker eine solche Einrichtung nicht.
Ein Verfahren wie in 2.3.2.1 beschrieben ist ebenfalls anwendbar. Es ist zwar etwas umständlicher, aber es funktioniert.
Die dritte und eleganteste Möglichkeit ist die Nutzung der

Formatierroutine. Hier brauchen Sie lediglich die Lage der Etiketten auf dem Träger einmal in fo$ zu hinterlegen, und los geht es. Für einen zweispaltigen Träger könnte das Programm so aussehen:

```
100fo$="aaaaaaaaaaaaaaaaaaaa        aaaaaaaaaaaaaaaaaaaa"
:
200a1$="Anrede1":a2$="Anrede2"
210fi$=a1$:gosub48000:fi$=a2$:gosub48000
220print#1,fp$
:
```

Auf die gleiche Weise können Sie nun mit den weiteren Zeilen fortfahren.

Der Vorteil liegt klar auf der Hand: Wie auch immer die Etiketten angeordnet sein mögen, sie brauchen nur fo$ entsprechend anzupassen, und der Drucker schafft bei nur einer Papierbewegung gleich mehrere Etiketten.

Ein kleiner Tip:

Übergeben Sie ausnahmsweise einmal für eine Zeile weniger Felder, als gemäß fo$ der Fall sein sollte, müssen Sie für die nächste Zeile fz%=0 setzen, damit fo$ wieder von vorne abgetastet wird.

Damit wären wir am Ende des Kapitels 2, und wir hoffen, daß wir Ihnen die drei Dinge, aus denen nach verbreiteter Auffassung (man munkelt) ein Programm bestehen sollte, in angenehmer und verständlicher Form auseinandergesetzt haben.

Unsere Zielsetzung war die, Ihnen in erster Linie brauchbare Werkzeuge an die Hand zu geben und dennoch, soweit es das Thema zuließ und es in dem jeweiligen Zusammenhang sinnvoll erschien, die Grundlagen zu diskutieren, um Sie zu eigenen Versuchen zu ermutigen.

Wir sind uns natürlich darüber im klaren, daß einige Programmteile schneller und vorteilhafter in Maschinensprache zu lösen gewesen wären, was jedoch nicht jedermanns Sache ist. Nicht alle Leser hätten etwas davon gehabt.

Vielleicht lassen Sie einmal von sich hören, wenn Sie

glauben, ein delikates Thema besonders elegant gelöst zu haben.
Auch wir lernen gerne hinzu (wir müssen es sogar).

3. KAPITEL: TIPS FÜR DIE PROFESSIONELLE PROGRAMMGESTALTUNG

In diesem Kapitel zeigen wir Ihnen anhand ausgewählter, wichtiger Themenbereiche, wie Sie Ihre Programme noch bediener- und anwenderfreundlicher gestalten können.

3.1 Parameterisierung von Programmen

Nehmen wir an, Sie haben eine Adreßverwaltung geschrieben, mit der Sie Adressen eingeben, ändern, suchen und löschen können. In Ihrem Programm haben Sie Länge des Datensatzes sowie die Anzahl der Felder und ihre Längen und Positionen verankert. Wenn Sie jetzt plötzlich feststellen, daß das Feld für den Namen zu kurz ist, geht die Sucherei los: Sie müssen sämtliche Stellen im Programm suchen, an denen die Länge dieses Feldes benutzt wird und dort entsprechend ändern. War dieses Feld nicht zufällig das letzte, so ändern sich selbstverständlich auch die Positionen der nachfolgenden Felder. Wie können wir solch eine umständliche Prozedur bei der Änderung eines Programmes vermeiden?

Das Stichwort hierzu heißt Parameterisierung. Für den Programmierer bedeutet dies, daß er anstelle von Konstanten für alle programmspezifischen Werte Variablen benutzt. Diesen Variablen werden zu Beginn des Programms einmal die Werte dieser speziellen Anwendung zugewiesen, die sogenannte Initialisierung. Dies kann z.B. durch ein Unterprogramm geschehen. Initialisieren kann man die Variablen entweder durch einfache '=' Anweisungen oder auch durch READ-Anweisungen, die die Werte einfach aus DATA-Statements holen. Sehen wir uns als Beispiel einmal einen Programmabschnitt an, der aus mehreren Variablen einen Datensatz zusammensetzt.

```
100 D$ = A1$ + A2$ + A3$ + A4$
110 PRINT# 1, D$
```

Parameterisiert sieht das Ganze z.B. so aus:

```
100 GOSUB 1000 :REM INITIALISIERUNG
110 D$ = ""
120 FOR I=1 TO N:REM ANZAHL DER VARIABLEN
130 D$ = D$ + A$(I) : NEXT
140 PRINT# 1, D$
...
1000 N=5 : DIM A$(N) : RETURN
```

Auf den ersten Blick sieht die zweite Variante umständlicher aus. Diese Version hat jedoch den Vorteil, daß bei einer Erweiterung der Anzahl der Variablen nur der Wert von N im Unterprogramm zur Initialisierung geändert werden muß.

Welche Werte sollten nur parametrisiert werden? Um beim Beispiel unserer Adreßverwaltung zu bleiben wären dies z.B. die Anzahl der Felder innerhalb des Datensatzes sowie die Länge jedes Feldes. Daraus kann das Programm dann die Länge des kompletten Datensatzes berechnen. Die Bezeichnungen der einzelnen Datenfelder könnten Sie in DATA-Statements ablegen und bei der Initialisierung einlesen, z.B.

```
100 READ N :REM ANZAHL DER DATENFELDER
110 DIM P(N+1),L(N) :REM POSITION, LÄNGE
120 P(1) = 1 : REM STARTPOSITION
130 FOR I=1 TO N
140 READ L(I) : REM LÄNGE LESEN
150 S = S + L(I) : REM GESAMTLÄNGE BRECHNEN
160 P(I+1) = P(I) + L(I) :REM POSITION BERECHNEN
170 NEXT
...
1000 DATA 5 : REM ANZAHL DER FELDER
1010 DATA 25, NAME
1020 DATA 20, VORNAME
1030 DATA 4, PLZ
1040 DATA 25, ORT
1050 DATA 20, STRASSE
```

Hier geben Sie also Anzahl, Länge und Feldbezeichnungen vor.

Grundsätzlich sollten Sie alle Konstanten in einem Programm zu Beginn als Variablen definieren. Dadurch werden Änderungen und Anpassungen in Ihrem Programm zum Kinderspiel. Sie können auf diese Weise z.B. aus Ihrer Adreßverwaltung eine Verwaltung für Schallplatten machen. Sie brauchen nur die entsprechende Initialisierungsroutine zu ändern. Diese Methode hat jedoch noch einen Nachteil. Sie können mit ein und dem selben Programm immer nur eine Art von Datei verwalten. Zur Anpassung an eine andere Aufgabenstellung müssen Sie das Programm ändern. Auch dies läßt sich vermeiden. Dazu legen wir die Parameter nicht im Programm, sondern in einer Datei ab. Wenn Sie nun das Programm benutzen, brauchen Sie nur per INPUT den Namen der Parameterdatei abfragen und die entsprechenden Daten von Diskette laden. Aufgrund dieser Daten können Sie z.B. Ihre Felder dimensionieren und die Eingabemasken mit den entsprechenden Texten aufbauen. Unser obiges Beispiel sähe dann so aus:

```
100 OPEN 2,8,2, "0:PARAMETER,S,R"
110 INPUT# 2,N :REM ANZAHL DER DATENFELDER
120 DIM P(N+1),L(N) :REM POSITION, LÄNGE
130 P(1) = 1 : REM STARTPOSITION
140 FOR I=1 TO N
150 INPUT# 2,L(I) : REM LÄNGE LESEN
160 S = S + L(I) : REM GESAMTLÄNGE BRECHNEN
170 P(I+1) = P(I) + L(I) :REM POSITION BERECHNEN
180 NEXT
190 CLOSE 2
```

Nach dem oben beschriebenen Verfahren lassen sich natürlich auch leicht Standardprogramme parameterisieren. Alle anwederspezifischen Daten werden nur einmal bei der Einrichtung des Programms eingegeben und in einer Parameterdatei auf Diskette gespeichert. Nach jedem Start liest das Programm zunächst die Parameterdatei und weist den zugehörigen Variablen im Programm die entsprechenden Werte

bzw. Texte zu. So entsteht aus dem Standard-Finanzbuchhaltungspaket XY schnell die individuelle Finanzbuchhaltung der Firma Maier & Söhne mit auf die eigenen Belange angepaßtem Kontenrahmen und indivduellen Mahntexten.

Bei der Parameterisierung gibt es jedoch noch eines zu beachten:

Da Sie in Ihrem Programm nicht wissen, welche Werte die Parameter später einmal erhalten, müssen Sie die erhaltenen Werte erst einmal auf Plausibilität prüfen. Dabei gilt es auch Sonderfälle zu berücksichtigen, z.B. bei Schleifen. Ist dort der Endwert kleiner als der Anfangswert, so wird die Schleife immer einmal durchlaufen, was evtl. nicht beabsichtigt war.

Was kann man nun alles parameterisieren?
Hier können Sie selbst entscheiden, wie weit Sie die Parameterisierung treiben wollen. Sie sollten jedoch nach Möglichkeit soviel wie möglich in Variablen packen. Nehmen wir als Beispiel mal unsere Adreßverwaltung. Hier sollten wir die Anzahl der Felder in unserem Datensatz variabel halten. Ebenso sollten die Längen der einzelnen Felder als Parameter behandelt werden. Natürlich gehören dann auch die Bezeichnungen der Felder dazu. Damit haben wir aus unserer Adreßverwaltung schon eine universelle Dateiverwaltung gemacht, mit der man z.B. auch seinen Bücherbestand verwalten könnte.

Bei der Parameterisierung sollten Sie auch die Peripheriegeräte Ihres Rechners berücksichtigen. Wenn Sie als Parameter den Druckertyp hinterlegen, können Sie im Programm entsprechend reagieren und Ihr Programm kann mit verschiedenen Druckertypen zusammen arbeiten.

In diesem Zusammenhang sollten Sie auch für Geräteadressen, Sekundäradressen und evtl. Filenamen Paramater verwenden. Benutzen Sie auch für die logische Filenummer einen Parameter, so können Sie damit auch Drucker mit und ohne

automatischem Zeilenvorschub bedienen. Ist die logische Filenummer größer als 127, so wir nach jedem Zeilen (Carriage Return, CHR$(13)) ein Line Feed (CHR$(10)) gesendet. Bei logischen Filenummern unter 128 unterbleibt dies.

Auch wenn für ein voll parameterisiertes Programm der Aufwand beim erstmaligen Erstellen größer ist, sollten Sie diesen Mehraufwand in Kauf nehmen. Er ist weitaus geringer als wenn Sie später Programm in allen Einzelheiten durchforsten müssen, um vielleicht das Feld für den Namen nur um ein Zeichen größer zu machen.

3.2 Menüsteuerung und Overlaytechnik

Eine bewährte Programmiertechnik besteht darin, für eine Problemlösung ein sogenanntes Menü- oder Auswahlprogramm zu schreiben, von dem aus für die einzelnen Teilprobleme jeweils ein eigenes Programm geladen und ausgeführt wird. Dieses Nachladen oder überlagern von Programme nennt man Overlaytechnik.
Ein kleines Menüprogramm kann z.B. so aussehen:

```
100 REM MENUE
110 READ N :REM ANZAHL DER PROGRAMME
120 FOR I=1 TO N
130 READ A$(I) : PRINT I, A$(I) : REM NAME UND NUMMER
140 PRINT : NEXT
150 PRINT "BITTE WÄHLEN SIE: "
160 GET T$ : T=VAL(T$) : IF T < 1  OR T   N THEN 160
170 LOAD A$(T),8 :REM PROGRAMM LADEN
180 DATA 3
190 DATA EINGABE, AUSWERTUNG, DRUCKAUSGABE
```

Das Programm liest zuerst die Anzahl der auszuwählenden Programme und zeigt diese dann mit der Nummer an. Wenn Sie jetzt eine Nummer eingeben, wird das entsprechende Programm geladen und automatisch gestartet. Ist das Programm beendet, wird von dort das Menüprogramm wieder geladen. Das ist die grundsätzliche Vorgehensweise. Dabei gibt es jedoch noch einen Punkt zu beachten:
Dies betrifft die Größe des aufrufenden und des aufgerufenen Programms sowie die übernahme der Variablen. Dabei gibt es zwei grundsätzliche Möglichkeiten: übernahme oder keine übernahme der Variablen in das nachgeladene Programm. Eine übernahme der Variablen ist nur dann möglich, wenn das aufrufende Programm mindestens so groß oder größer als das nachgeladene ist. Wird von einem Programm aus ein anderes Programm nachgeladen, so bleiben die Zeiger auf das Programmende erhalten und das neue Programm wird vom Beginn an abgearbeitet. In unserem Beispiel würden wir folgendes Ergebnis erhalten:

```
100 REM PROGRAMM 1
110 REM DIESES PROGRAMM IST GRöSSER ALS DAS ZWEITE
120 A = 1000
130 LOAD "PROGRAMM 2",8
```

```
100 REM PROGRAMM 2
110 PRINT A
```

```
1000
```

Ist das nachgeladene Programm jedoch größer als das ursprüngliche Programm, so würde ein Teil der Variablen überschrieben und wir erhielten undefinierte Werte. Außerdem würde bei Wertzuweisungen an Variablen der Teil des Programms zerstört, der über die Länge des ersten Programms hinaus geht.

Beim übernehmen vom Variablen gibt jedoch noch zwei Besonderheiten zu beachten: Handelt es sich um Stringvariablen, die im ersten Programm als Konstanten in Anführungszeichen definiert worden sind, so gibt es Probleme. Bei Stringvariablen wird ein Zeiger verwendet, der auf den eigentlichen Text der Variablen zeigt. Wird eine Stringvaribable nun z.B. mit folgender Anweisung im ersten Programm definiert

```
100 A$ = "TEXT"
```

so zeigt der Variablenzeiger in den Programmtext. Beim Nachladen des nächsten Programms wird nun dieser Zeiger nicht verändert. An der ursprünglichen Stelle steht jetzt jedoch der neue Programmtext, so daß die Variable nun einen undefinierten Inhalt hat. Dies können wir jedoch leicht umgehen. Wir brauchen bloß dafür zu sorgen, daß der Text aus dem Programm in den oberen RAM-Bereich kopiert wird, in dem die Textvariablen normalerweise stehen. Dies erreichen wird z.B. durch folgende Programmzeile:

```
100 A$ = "TEXT" + ""
```

Durch die Addition des Leerstrings wird das Kopieren des Variableninhalts in den Stringbereich erzwungen. Ähnliche Überlegungen gelten auch bei Funktionsdefinitionen, da auch hier der Zeiger auf die Definition im Programm zeigt. Hier müssen wir die Funktion in zweiten Programm noch einmal definieren, z.B.

```
100 DEF FN A(X) = 0.5 * EXP (-X*X)
```

Halten wir noch einmal fest:
Wollen wir ein Programm nachladen, so können wir die Variablen nur dann weiter benutzen, wenn das zweite Programm kleiner als das erste Programm ist. Ist das nachgeladene Programm größer und sollen keine Variablen übernommen werden, können wir uns mit einem Trick aus der Affäre ziehen:

Wir brauchen lediglich unmittelbar nach dem Laden den Zeiger auf das Ende des BASIC-Programms auf den Wert des neuen Programms setzen. Dies ist mit zwei POKE-Befehlen möglich, da die Endadresse nach dem Laden zur Verfügung steht:

```
POKE 45, PEEK(174) : POKE 46, PEEK(175) : CLR
```

Der CLR-Befehl ist unbedingt erforderlich. Diese Zeile sollte als erste im nachgeladenen Programm stehen. Damit haben wir also die Möglichkeit geschaffen, beliebig große Programme ohne Variablenübergabe nachzuladen. Diese Methode werden wir sicher auch bei Menütechnik wählen. Die Datenübergabe könnte dann z.B. auch über Dateien auf Diskette geschehen.
Eine andere nicht so elegante Möglichkeit besteht darin, den Ladebefehl in den Tastaturpuffer zu schreiben und das Programm dann im Direktmodus automatisch nachladen zu können. Dazu schreiben wir vor dem Laden den LOAD- und RUN-Befehl auf den Bildschirm und füllen den Tastaturpuffer mit 'HOME' und Carriage Return. Im Programm muß danach eine END-Anweisung stehen. Das Betriebssystem holt dann im Direktmodus den Inhalt des Tastaturpuffers und liest damit

den LOAD- und RUN-Befehl, der zum Laden und Ausführen des Programms führt. Da dies im Direktmodus geschieht, werden automatisch die Endadresse des Programms gesetzt, die Variablen gelöscht und mit dem nachfolgendem RUN das Programm gestartet. Der Nachteil hierbei ist jedoch, daß der Ladebefehl auf den Bildschirm geschrieben und eine evtl. Bildschirmmaske dabei zerstört wird. In der Praxis sähe das so aus:

```
....
1000 PRINT CHR$(147)"LOAD"CHR$(34)"PROGRAMM 2"CHR$(34)",8"
1010 PRINT : PRINT : PRINT : PRINT
1020 PRINT "RUN"
1030 POKE 631,19 : POKE 632,13 : POKE 633,13
1040 POKE 634,13 : POKE 635,13 : POKE 636,13
1050 POKE 198,6  : END
```

Sie sehen schon, daß dieses Verfahren umständlicher als das oben geschilderte Verfahren ist; es ist nur der Vollständigkeit halber erwähnt. Beim ersten Verfahren wäre nur in Zeile 1000 der programmierte LOAD-Befehl erforderlich:

```
1000 LOAD "PROGRAMM 2",8
```

und im aufgerufenen Programm die beiden POKE-Befehle und der CLR.

3.3 Mehr Programmkomfort durch Funktionstasten

Auf der rechten Seite Ihres Commodore 64, griffgünstig angeordnet und schön groß, finden Sie die sogenannten Funktionstasten mit den Nummern 1 - 8

Vielleicht haben Sie sich schon oft gefragt, warum diese Tasten existieren und wie man sie anwendet. Darauf wollen wir Ihnen in diesem Kapitel erschöpfende Antwort geben.

WARUM Funktionstasten ?

Funktionstasten haben die günstige Eigenschaft, daß Sie eigentlich keinerlei Bedeutung besitzen. Dies mag sich zuerst einmal etwas komisch anhören, erhält jedoch seinen Sinn durch die dadurch gegebene, völlig freie Programmierbarkeit.

Dadurch ist es möglich, Funktionstasten eine freie Belegung zu geben. Dabei sollte man als Programmierer aber ein paar Spielregeln beachten.

Zum einen ist es zwar eine wunderbare Sache, jede beliebige Funktion auf diese Tasten legen zu können, zum anderen ist es aber auch für den Anwender nicht ganz einfach, sich diese Funktionen zu merken. Aus diesem Grund sollte man sich genau überlegen, welche Funktionen auf die Tasten gelegt werden.
Zum zweiten sollten bestimmte Funktionen im gesamten Programm, besser noch in allen Programmen, die Sie schreiben, die gleiche Bedeutung haben.

Nehmen wir als Beispiel die beiden Programme TEXTOMAT und DATAMAT aus unserem Hause. Beide Programme machen regen Gebrauch von den Funktionstasten. Zwei der Tasten haben jedoch immer, d.h. in jedem Programmteil und zu jedem Zeitpunkt, die gleiche Bedeutung. Die Taste F1 bedeutet immer, daß eine Eingabe abgeschlossen ist oder ein Befehl ausgelöst wird. Die Taste F2 bedeutet immer, daß die Arbeit

in einem Programmteil abgebrochen wird. Mit dieser Taste können Sie jeden Programmteil jederzeit verlassen.

Es ist von enormen Vorteil, daß der Anwender immer die gleiche Taste für diese Funktionen betätigen muß. Dadurch kann er ohne Umstellungsschwierigkeiten von einem Programm zu einem anderen umsteigen.

Doch warum nehmen wir eigentlich ausgerechnet die Funktionstasten ?

Einen Vorteil, die freie Belegbarkeit, haben wir bereits oben erwähnt. Es ist erheblich einfacher, auch vom Programm her, die Taste F1 abzufragen als die Taste "f" für fertig. Diese Taste wird ja auch zur normalen Texteingabe benötigt.

Da die Funktionstasten für die normale Arbeit keinerlei Bedeutung haben, sind sie auch jederzeit abfragbar, ohne daß der normale Programmablauf gestört wird. Im Prinzip könnten Sie nach jeder Programmzeile zu einem Unterprogramm verzweigen, das die Funktionstasten abfragt. Inwieweit dies sinnvoll ist, ist natürlich ein anderes Problem.

Ein weiterer erheblicher Vorteil ist die Bedienung der Tasten selber. Die Tasten sind griffgünstig auf der rechten Seite angeordnet und schön groß ausgefallen. Dadurch sind sie leicht zu erreichen und man kann eine Fehlbedienung fast gänzlich ausschließen.

Nachdem wir nun geklärt haben, warum Funktionstasten zu benutzen sind, wenden wir uns nun der Frage zu, WIE die Abfrage zu bewerkstelligen ist.

Das große Geheimnis läßt sich ganz einfach lösen. Die Funktionstasten haben wie alle anderen Tasten auch einen bestimmten Zahlencode, der mittels einer einfachen GET-Schleife abgefragt werden kann.

Die Codes sind die Folgenden :

F1 - 133
F2 - 137
F3 - 134
F4 - 138
F5 - 135
F6 - 139
F7 - 136
F8 - 140

Diese Werte brauchen Sie in Ihrem Programm nur abzufragen. Dies könnte zum Beispiel so aussehen

```
10 GET E$ : IF E$=CHR$(133) THEN 1000 : REM F1
20 IF E$=CHR$(137) THEN 9999 : REM F2
30 GOTO 10
.
.
.
1000 REM Hier geht das Programm los
.
.
.
9999 END : REM Hier ist das Programm zu Ende
```

In Zeile 10 wird die Funktionstaste F1 abgefragt. Laut Tabelle ist der Code für F1 133. Wenn F1 gedrückt ist, so hat E$ diesen Wert. Dies wird abgefragt und entsprechend im Programm verzweigt. In Zeile 20 wird genauso die Funktionstaste F2 abgefragt.

Sie sehen, daß das Problem der Funktionstastenabfrage eigentlich ganz einfach zu lösen ist. Sie fragen diese Tasten einfach wie alle anderen Tasten ab.
Die universelle Verwendbarkeit ergibt sich dann daraus, daß diese Tasten von Haus aus mit keiner Bedeutung belegt sind.

3.4 Testen Sie ihr Programm aus - Anwender sind keine Versuchskaninchen

Nichts ist wohl in der Programmierung schlimmer, als ein Anwender, der Sie laufend telefonisch mit neuen Fehlermeldungen beglückt, und Sie dann irgendwann vielleicht sogar mit seinem Anwalt bedroht. Das muß und darf nicht so sein ! Professionelle Programme sollen, bevor sie in die Hand des späteren Anwenders gelangen, sorgfältig ausgetestet werden. Was zunächst nach lästiger Mehrarbeit aussieht, ist in Wirklichkeit eine wichtige Arbeitserleichterung. Dies gilt natürlich insbesondere dann, wenn Sie ein und dasselbe Programm mehreren Anwendern zur Verfügung stellen.

Der Vorgang des Testens beginnt sinnvollerweise bereits bei der Programmerstellung. Testen Sie jeden Programmteil und jedes Unterprogramm nach Fertigstellung zunächst isoliert mit willkürlich gewählten Daten und dann erst im Programmzusammenhang. So ersparen Sie sich später eine langwierige Fehlersuche. Der Vorteil einer strukturierten Programmierweise und die Arbeit mit erprobten Bausteinen kommen hierbei voll zur Geltung. Sehr hilfreich für das Austesten von Programmen sind die im COMMODORE 64 Befehlssatz leider nicht vorhandenen sogenannten TOOLKIT-Befehle, wie z.B. TRACE, FIND und DUMP. Besonders der Befehl TRACE, der die jeweiligen Programmbefehle bei der Ausführung auf dem Bildschirm anzeigt, erweist sich als sehr nützlich. Die TOOLKIT-Funktionen sind in den unterschiedlichen zum COMMODORE 64 erhältlichen Befehlserweiterungen enthalten. Achten Sie bei der Auswahl der entsprechenden Befehlserweiterungen unbedingt darauf, daß die TRACE-Funktion wahlweise auch Einzelschrittverarbeitung zuläßt.
Sehr wesentlich ist das Erstellen der Dokumentation paralell zur Programmierung. Eine erst nach Programmfertigstellung verfaßte Dokumentation weist nicht nur oft Vergeßlichkeitslücken auf, sondern ist schon fast Verschwendung. Bei der Fehlersuche, aber auch bei der

Verhinderung der Fehlerentstehung ist die Dokumentation ein sehr wesentliches Hilfsmittel. Denken Sie z.B. einmal an den häufigen und typischen Fehler der Mehrfachdefinition von Variablen. Mit einer ordentlich geführten Variablenliste kann soetwas nicht passieren. Auch Dateibeschreibungen und Bildschirmmasken sollten vor der eigentlichen Programmierung erstellt werden. Für viele Programmierer ist leider das Erstellen einer Dokumentation eine lästige Pflichtübung, die lange nach Programmerstellung mehr schlecht als recht vorgenommen wird. Gewöhnen Sie sich diese Unart bitte gar nicht erst an !

Bei hartnäckigen Fehlern, die auch mit intensiver Testarbeit manchmal nicht oder nur sehr schwer zu finden sind, kann es durchaus ratsam sein, die entsprechende Problemstellung noch einmal anders programmtechnisch zu lösen. Leider kann es auch immer wieder vorkommen, daß ein solcher unerklärlicher Fehler auf eine Unzulänglichkeit im Betriebssystem des COMMODORE 64 oder seiner Peripherie zurückzuführen ist. Da helfen dann nur Programmänderungen Ihrerseits, die derartige Fehlerquellen umgehen.

Noch ein anderer Punkt sollte beim Testen, vor allem des fertigen Programms, berücksichtigt werden. Hierzu ein aktuelles Beispiel: das in diesem Buch dargestellte Dateizugriffsverfahren QUISAM wurde von Ersteller Klaus Gerits nach allen Regeln der Kunst getestet und für fehlerfrei befunden. Damit wäre es beinahe in der ursprünglichen Version in dieses Buch gekommen. Doch wir hatten (geplantes) Glück. Eins der im letzten Teil des Buches vorhandenen Anwendungsprogramme sollte den Gebrauch von QUISAM illustrieren. So schrieb Wolfgang Schellenberger auf Basis von QUISAM eine Literaturverwaltung. Und siehe da - er brachte sein Programm zunächst nicht ans Laufen, da in QUISAM doch noch zwei kleine Fehler steckten. Diese waren bisher unentdeckt geblieben, da sie nur bei ganz bestimmten Konstellationen auftraten. Aus eigener Erfahrung können wir Ihnen deshalb nur empfehlen, Programme nicht nur selbst ausführlich und eingehend zu testen, sondern sie auch noch

einem oder mehreren unvoreingenommenen Dritten zum Test zu geben. Auch bei der Programmierung gibt es das Problem der "Betriebsblindheit", wodurch manchmal sogar gravierende Fehlermöglichkeiten nicht erkannt werden.

Vorsicht ist auch geboten bei nachträglichen Änderungen und Verbesserungen an fertigen Programmen. So manch ausgereiftes, fehlerfreies Programm wurde aufgrund eines neuen Geniestreiches seines Erstellers wieder zur ärgerlichen Fehlerquelle. Klären Sie deshalb vor einer Programmänderung anhand von Dokumentation und Variablenliste sorgfältig ab, auf welche Programmteile diese Änderung alles Einfluß nimmt. Natürlich muß auch jede Änderung wieder entsprechend dokumentiert werden. So können Sie dann unliebsame Überraschungen weitgehend vermeiden. Vor steten Änderungen eines von Dritten benutzten Programms sei ohnehin gewarnt. Erheblich sinnvoller ist es, in größeren Zeitabständen komplette Neufassungen Ihres Programms herauszugeben, die dann in ausgetesteter Form alle Änderungen enthalten. Auch der Anwender Ihres Programms wird es schätzen, wenn er sich nicht laufend umgewöhnen muß

3.5 Die Bedienungsanleitung - das Aushängeschild eines Programmes

Viele Programmierer schreiben zwar ganz hervorragende Programme, vergessen dann aber leider, daß diese Programme auch für andere, normalsterbliche Menschen zugänglich sein sollen. Daß dies anhand einer guten und ausführlichen Dokumentation und Bedienungsanleitung geschieht scheinen die wenigsten bisher gehört zu haben.

Ebenfalls zu beachten ist, daß der Käufer eines Programmes in 99% aller Fälle zuerst einmal die Bedienungsanleitung durchliest und der erste Eindruck vom Programm also aus der Bedienungsanleitung resultiert. Sehr oft werden besonders preiswerte Programme nicht im Geschäft vorgeführt, so daß der Käufer die Kaufentscheidung einzig und allein von der Bedienungsanleitung abhängig macht. Dieses Kriterium ist unter kommerziellen Gesichtspunkten sicher nicht zu vernachlässigen.

Doch neben dem geschäftlichen Aspekt bleibt auch zu berücksichtigen, daß eine gute Bedienungsanleitung dem Programmierer auch viele Rückfragen seitens der Anwender erspart.

Da Sie nun hoffentlich erkannt haben, wie immens wichtig eine gute Bedienungsanleitung ist, fragen Sie sich sicher, wie eine solche auszusehen hat.

Dazu müssen wir erst einmal klären, welchen Anforderungen eine Bedienungsanleitung genügen muß.

Zuerst einmal muß eine Bedienungsanleitung absolut für den Anfänger geschrieben sein. Wenn Sie eine solche schreiben, müssen Sie sich grundsätzlich vorstellen, daß ein absoluter Laie sich vor den Computer setzt und versucht mit Ihrem Programm zu arbeiten.
Dies bedeutet, daß Sie zuerst einmal alle im Text

verwendeten Fachbegriffe erklären müssen. überhaupt sollten Sie sich davor hüten, allzuviele Fachbegriffe in den Text aufzunehmen. Manches läßt sich sicher nicht vermeiden, aber beachten Sie immer, daß für Sie als Profi vieles selbstverständlich ist, was für Anfänger absolut nicht zu verstehen ist. Wie gesagt, sollten Sie ALLE Fachbegriffe zu Beginn des Handbuches erklären.

Neben der leichten Verständlichkeit ist auch eine klare und übersichtliche Gliederung des Handbuches von enormer Wichtigkeit. Dazu gehören ein ausführliches Inhaltsverzeichnis, ein alphabetisches Stichwortverzeichnis und eine saubere thematische Gliederung des Textes. Der Anwender muß in der Lage sein, ohne großen Aufwand jede aufkommende Frage klären zu können.

Der dritte wichtige Faktor ist, daß Sie immer mit Ermüdungserscheinungen des Lesers rechnen müssen. Um dem entgegenzuwirken sind neben den vielen selbstverständlichen Beispielen auch Grafiken und Bildschirmausdrucke nicht zu vernachlässigen. Die Optik eines Bedienungshandbuches wird hauptsächlich von solchen auffälligen Grafiken bestimmt. Dies bedeutet, daß diese natürlich sauber und exakt sein müssen und aber auch den Leser anregen sollen.
Rechnen Sie immer damit, daß der Leser ermüdet. Kleine Erfolgserlebnisse oder Grafiken, die sofort ins Auge fallen, wirken diesem Phänomen sehr gut entgegen.

Doch wie bauen wir nun ein solches Handbuch auf?

Im folgenden stellen wir Ihnen einen bewährten und erfolgreichen Aufbau vor, der natürlich keinerlei Anspruch auf Vollständigkeit erhebt und auch sicher nicht unbedingt der Weisheit letzter Schluß ist. Es hat sich aber gezeigt, daß gerade Anfänger mit diesem Aufbau ganz hervorragend zurechtkommen.

Im Groben gesehen besteht bei dem vorgeschlagenen Aufbau das Handbuch aus zwei Teilen. Im ersten Teil, dem sogenannten Übungshandbuch, wird der Anwender schrittweise anhand von Beispielen in die Arbeit mit dem Programm eingeführt.

Dies beginnt mit der Erläuterung der wichtigsten Fachbegriffe und beschreibt zuerst einmal, wie das Programm gestartet wird. Dieser enorm wichtige Punkt findet sich zwar in fast allen Handbüchern, doch meistens nicht am Anfang. Zur Startprozedur gehört auch, wie der Rechner und seine Peripherie eingeschaltet wird, wie die Diskette einzulegen ist und wie das Programm anschließend gestartet wird. Dabei ist jeder Tastendruck zu erklären, z.B.:

> Legen Sie die Diskette in Ihr Laufwerk ein, schließen Sie die Laufwerkstür und tippen Sie ein:
>
> LOAD"*",8
>
> Betätigen Sie nun die Taste mit der Aufschrift "RETURN", im folgenden einfach "RETURN"-Taste genannt.
>
> Nach ein paar Sekunden blinkt der Cursor wieder auf dem Bildschirm. Tippen Sie nun ein :
>
> RUN
>
> Betätigen Sie wieder die "RETURN"-Taste.

Auch wenn Ihnen diese haarkleine Erklärung jedes einzelnen Arbeitsschrittes etwas überflüssig vorkommt, so bedenken Sie, daß Sie damit rechnen müssen, daß absolute Laien das Programm bedienen.

Nach der Startprozedur können Sie im Handbuch ein paar

allgemeine Hinweise zu dem Programm geben. Dazu gehört, wie das Programm arbeitet und was eventuell besonders zu beachten ist.

Dann sollten Sie sofort anhand eines einfachen Beispieles in die Arbeit einsteigen. Es empfiehlt sich, daß Übungshandbuch in mehrere Lektionen einzuteilen, entsprechend dem Vorgehen im Programm.

Es ist nicht Sinn des Übungshandbuches, dem Anwender in alle Feinheiten und Tricks einzuführen, sondern er soll alle wichtigen (aber auch nur die) Teile kennenlernen und damit umgehen können. Dabei gilt es zwei Dinge zu beachten. Zum einen muß alles absolut narrensicher beschrieben sein, wozu auch gehört, daß auf eventuell mögliche Fehlbedienungen hingewiesen wird und wie Abhilfe zu schaffen ist. Zum zweiten ist es ungeheuer wichtig, daß der gesamte Stoff in kleinen und leicht verdaulichen Happen mitgeteilt wird. Der Anwender muß immer wieder kleine Erfolgserlebnisse haben. Es gibt nichts besseres wie ein Bildschirmdruck im Handbuch, den der Anwender genauso auch auf seinem Schirm hat. In diesem Fall freut sich jeder, daß er alles richtig gemacht hat und kommt gar nicht auf die Idee, mit der Arbeit aufzuhören. Eine gute Bedienungsanleitung erkennt man daran, wie lange jemand ununterbrochen damit arbeitet.

Noch einmal zusammenfassend :

Das Übungshandbuch soll den Anwender schrittweise in die Arbeit mit dem Programm einführen. Dabei kann auf die Feinheiten verzichtet werden. Wichtig ist, daß viele Beispiele verwendet werden, der Anwender immer wieder kleine Erfolgserlebnisse hat und das Ganze absolut leicht verständlich ist.

Der zweite Teil des Handbuches besteht aus dem sogenannten Anwenderhandbuch. Dieses enthält nach Themen geordnet eine

Beschreibung aller Programmteile. Zu dieser Beschreibung gehört, wie man überhaupt in diese Teile gelangt, wozu sie dienen, wie sie bedient werden, was man falsch machen kann, wie man sie wieder verläßt.

Wichtig ist, daß in diesem Teil des Handbuches wirklich jedes Detail erklärt wird. Nichts, und erscheint es auch noch so überflüssig, darf ausgelassen werden. Jeder denkbare Fall muß berücksichtigt und ausführlich erläutert werden. Eigentlich kann dieser Teil des Handbuches gar nicht umfangreich genug sein, doch gilt es auch dabei zu beachten, daß der Anwender sich durchfinden muß. Dazu dient neben einer einsichtigen und klaren Gliederung, daß trotz der geforderten Ausführlichkeit nichts überflüssiges und Verwirrendes sich im Text findet. Hier müssen Sie als Programmierer selber wissen, was notwendig ist und was nicht und einen guten Kompromiß finden.

Das Anwenderhandbuch sollte auch eine ausführliche Fehlerliste enthalten, die jede mögliche Fehlermeldung auflistet, die mögliche Ursache beschreibt und was dagegen zu tun ist.

Ein ausführliches Inhaltsverzeichnis gehört natürlich zu jedem guten Handbuch. Leider nicht immer selbstverständlich ist ein alphabetisches Stichwortverzeichnis. Dieses sollte alle wichtigen Begriffe, Programmnamen, Befehle und so weiter versehen mit der Seitennummer, unter der sich die Erläuterung befindet, enthalten.

Oft wünschenswert ist auch noch eine kurze Zusammenfassung aller wichtigen Befehle mit einer zu jedem Befehl nur ein paar Worte umfassenden Erläuterung. Die Notwendigkeit einer solchen Kurzreferenz ist aber vom Programm abhängig.

Fassen wir noch einmal zusammen :

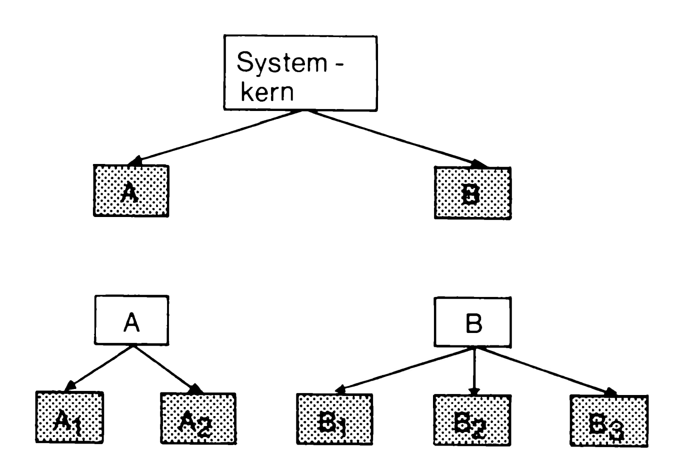
Wichtig ist zuerst einmal die gute Verständlichkeit des Handbuches. Dazu gehört die Erklärung aller Fachbegriffe. Wenn Sie Ihr Handbuch dann noch vernünftig gliedern, keinen Programmteil vergessen und Wert auf die nicht zu unterschätzende optische Gestaltung legen, steht einem guten Handbuch nichts mehr im Wege.

Sicher sind dies alles keine sehr leicht zu erfüllenden Forderungen, wie an den auf dem Markt befindlichen Bedienungsanleitungen oft zu ersehen ist. Doch ein Programm steht und fällt mit seiner Anleitungen, so daß Sie sich dafür einfach genügend Zeit nehmen müssen.

3.6 TOP-DOWN oder BOTTOM-UP ?

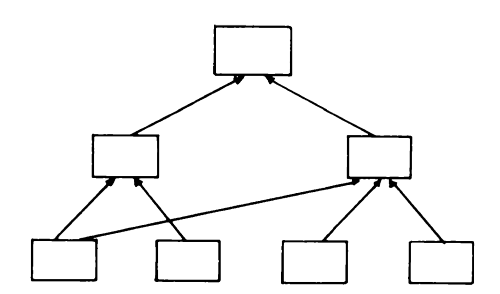
Bei großen Programmen stellt sich immer die Frage, in welcher Reihenfolge die verschiedenen Teilaufgaben erledigt werden sollen. Für die zwei einzig sinnvollen Methoden haben sich die Begriffe TOP-DOWN- und BOTTOM-UP-Programmierung eingebürgert.

TOP-DOWN-Programmierung bedeutet, daß ausgehend von einer groben Struktur das Programm nach und nach in den einzelnen Teilen fertiggestellt wird. Bei einer Dateiverwaltung sähe dies zum Beispiel so aus, daß man zuerst ein Hauptmenue programmiert, von dem aus alle Programmteile aufgerufen werden. Diese Programmteile bestehen zu Beginn erstmal nur aus einem Rücksprung.
Der nächste Schritt ist nun, sich in jedem Programmteil zu überlegen, wie diese zu untergliedern sind und weitere Unterprogramme aufzurufen. Durch diese Arbeitsweise reduziert man sein großes Gesamtproblem auf viele kleine und einfach zu lösende Einzelprobleme. Die schrittweise Verfeinerung wird an dem folgenden Diagramm recht deutlich.



Diese schrittweise Verfeinerung läßt sich besonders gut in den Vorarbeiten bei der Erstellung der Flußdiagramme oder Struktogramme anwenden.

Die BOTTOM-UP-Programmierung beginnt auf der untersten Ebene. Man schreibt häufig benutzte Programmteile, z.B. Lesen oder Entpacken eines Datensatzes zuerst einmal als Unterprogramm. Diese können dann von Programmteilen einer höheren Stufe wieder aufgerufen werden und diese selbst werden auch wieder aufgerufen. Im günstigsten Fall besteht das Hauptprogramm dann nur noch aus einer Reihe von Unterprogrammaufrufen.
Auch hierzu wieder ein Diagramm:



In der Praxis verwendet man eine Mischform aus den beiden Programmiertechniken. Zuerst wird man bei größeren Programmen den groben Ablauf festlegen und die erste Unterprogrammebene schaffen. Dann empfiehlt es sich, einzelne Module zu programamieren und auszutesten, auch um festzustellen, ob sich die Idee so überhaupt realisieren läßt.
In der Schlußphase trifft man sich dann in der Mitte mit einem hoffentlich fehlerfreien Programm.
Bei kleineren Programmen dagegen kann man meist direkt mit dem Hauptteil bzw. den einzelnen Routinen beginnen und das Programm dann immer weiter ausbauen.

3.7 Logische Felder - 8 Entscheidungen pro Byte

Oft haben Sie in einer Dateiverwaltung Eingabefelder, die nur eine Ja/Nein-Entscheidung verlangen. Zum Beispiel männlich/weiblich oder verheiratet ja/nein und so weiter.

In diese Felder, die auf einer Karteikarte oftmals angekreuzt werden, sind wie gesagt nur zwei Eintragungen zulässig. Aus diesem Grund wäre es mehr als unsinnig, ein ganzes Byte für eine einzige solche Entscheidung zu verschwenden.

Wie Sie sicher wissen, besteht ein Byte aus 8 Bits, von denen jedes entweder den Wert 1 oder 0 haben kann. Insgesamt können Sie also 8 Entscheidungen in einem solchen Byte unterbringen. Doch wie fassen wir dies nun programmtechnisch an.

Dazu müssen wir uns etwas der Boolschen Algebra bedienen. Es soll hier nun keine tiefe mathematische Abhandlung über diese folgen. Wer tiefer in dieses interessante Gebiet einsteigen möchte, sei auf die einschlägige mathematische Fachliteratur verwiesen. Uns interessieren an dieser Stelle nur die praktische Anwendung und Auswirkung. Wir benötigen dazu nur zwei Operationen, die AND- und die OR-Funktion.

Außerdem müssen wir noch die Wertigkeiten der 8 Bits pro Byte kennen. Beginnen wir damit :

128 64 32 16 8 4 2 1

Die acht Zahlen geben diese Wertigkeiten an. Sie erkennen schon, daß es die Zweierpotenzen von 0 bis 7 sind.

Um nun z.B. das erste Bit auf den Wert eins zu setzen, bedienen wir uns der OR-Funktion. Nehmen wir an, die Variable B soll die 8 Entscheidungen speichern. Um dem ersten Bit den Wert 1 zuzuweisen, oderieren wir die

Variable.

B = B OR 1

Damit ist das erste Bit gesetzt, egal ob es vorher gesetzt war oder nicht.

Nehmen wir an, B hat den Wert 127, dies sieht von der Bitstruktur her so aus:

11111110

Die oben angegebene Rechnung hätte zur Folge, daß das Bild nun so aussieht:

11111111

Die Variable B hat nun den Wert 255

Wollen Sie einem beliebigen Bit den Wert eins zuweisen, so nehmen Sie einfach die Wertigkeit dieses Bits und oderieren die Variable B damit. Noch ein Beispiel: B soll den Wert 163 haben :

11000011

Wir wollen nun das 5. Bit auf 1 setzen, dieses hat die Wertigkeit 16:

B = B OR 16

Danach sieht die Bitstruktur folgendermaßen aus :

11010011

B hat nun den Wert 163 + 16 = 179.

Ähnlich ist die Vorgehensweise beim Löschen eines Bits, d.h. das Bit soll den Wert 0 erhalten.
Allgemein läßt sie sich so beschreiben :

Berechne 255 minus der Wertigkeit des Bits und undiere die Variable B damit.

Dazu wieder ein Beispiel :

B soll den Wert 161 haben, Bit 6 soll gelöscht werden.

Die Bitstruktur sieht folgendermaßen aus :

10100001

Wir rechnen : 255 - 32 = 223

Nun wird undiert :

B = B AND 223

Wir erhalten :

10000001

B hat nun den Wert 161 - 32 = 129

Zum Schluß noch ein kurzes Programm, das den praktischen Einsatz dieser Möglichkeit Entscheidungen zu speichern, demonstriert.

Da die Variable B keinen Wert größer als 255 enthalten kann, ist es möglich, sie mit Hilfe des CHR$-Befehls in nur einem Byte abzuspeichern und somit sehr platzsparend unterzubringen.

3.8 Das Datum - der besondere Datentyp

Das Datum wird in fast jeder kommerziellen Anwendung benötigt, erfordert aber wegen seiner besonderen Struktur eine Spezialbehandlung.
Diese beginnt bei der Art der Speicherung, geht über das Sortieren und über die Berechnung eines bestimmten Wochentages nach beliebigem Datum. Alle drei Fälle wollen wir hier nacheinander behandeln.

Gehen wir davon aus, daß das Datum in folgender Form vorliegt:

Die Variable T enthält den Tag, die Variable M den Monat und J das Jahr. Die einfachste Art und Weise der Speicherung wäre nun, die Variablen einfach so abzuspeichern. Dies kann aber bis zu 8 Bytes pro Datum erfordern, je nach Art des Datums.
Durch die besondere Struktur des Datums ist es aber möglich, eine platzsparendere Art der Speicherung zu benutzen. Für den Tag sind nur Zahlen zwischen 1 und 31 zulässig, für den Monat zwischen 1 und 12, für das Jahr zwischen 0 und 99, wenn wir das Jahrhundert weglassen. Da keine dieser Zahlen die magische Grenze von 255 überschreitet, könnte man diese durchaus mit Hilfe des BASIC-Befehls CHR$ in Zeichen umwandeln.
Eine solche platzsparende Speicherung des Datums könnte folgendermaßen aussehen:

```
1000 DA$ = CHR$(J) + CHR$(M) + CHR$(T)
```

Damit haben Sie das Datum in nur 3 Bytes komplett abgespeichert.

Der umgekehrte Weg, das Datum wieder in die ursprüngliche Form bringen, ist auch ganz einfach :

```
2000 J = VAL(LEFT$(DA$,1))
```

```
2010 M = VAL(MID$(DA$,2,1))
2020 T = VAL(RIGHT$(DA$,1))
```

Vielleicht haben Sie sich gewundert, daß die Variable DA$ das Datum in der Reihenfolge Jahr, Monat, Tag speichert. Dies findet seine Begründung im Sortieren, das genau in dieser Reihenfolge stattfinden muß.

Durch diese Art der Speicherung ist es nämlich äußerst einfach geworden, nach dem Datum zu sortieren, da es nun wie jede andere Zeichenkette, zum Beispiel ein Name, behandelt werden kann.

Wenn Sie eine Liste von Daten haben, so können Sie diese nun in ein Array abspeichern und dann nach einem der bekannten Algorithmen sortieren (siehe Kapitel 2)

Oft stellt sich auch das Problem, nach einem vorgegebenen Datum den zugehörigen Wochentag zu berechnen. Das folgende Programm berechnet jeden beliebigen Wochentag des Gregorianischen Kalenders.

Das Programm ist als Unterprogramm ausgelegt. Es verlangt als Eingabeparameter den Tag in TA, den Monat in MO und das Jahr in JA. Wichtig ist, daß das Jahr vollständig, d.h. vierstellig angegeben wird, z.B. 1984.

Es liefert den Wochentag in T$ zurück. Ebenfalls kann in IN eine Zahl zwischen 0 und 6 abgefragt werden, die den Code für den Wochentag angibt.

```
100 REM WOCHENTAGSBERECHNUNG
110 INDEX = TAG
120 INDEX = INDEX + 31 * ( MO - 1 )
130 INDEX = INDEX + 365 * JAHR
140 IF MO > 2 THEN INDEX = INDEX - INT( 0.4 * MO + 2.3 ) : GOTO 160
150 JAHR = JAHR -1
160 INDEX = INDEX + INT ( JAHR / 4 )
170 X = INT ( JAHR / 100 ) + 1
180 INDEX = INDEX - INT ( 0.75 * X )
190 INDEX = INDEX - ( INT ( INDEX / 7 ) * 7 )
200 RESTORE
210 FOR J = 0 TO INDEX
220 READ T$
230 NEXT J
240 RETURN
300 DATA "SAMSTAG" , "SONNTAG" , "MONTAG" , "DIENSTAG" , "MITTWOCH"
310 DATA "DONNERSTAG" , "FREITAG"

READY.
```

3.9 Daten gepackt speichern - mehr als 255 Zeichen pro Datensatz

Beim COMMODORE 64 liegt die maximale Größe für einen Datensatz bei 255 Zeichen. Dies ist in der internen 8-Bit-Struktur des Rechners begründet.

Sollen mehr als 255 Zeichen pro Datensatz gespeichert werden, so muß diese physikalische Grenze irgendwie umgangen werden.

Dazu gibt es zwei Möglichkeiten :

1.) Die physikalische Grenze wird logisch umgangen. Dies bedeutet, daß zwei oder mehr physikalische Datensätze zu einem logischen Datensatz zusammengefaßt werden. Diese Vorgehensweise empfiehlt sich besonders, wenn viel Text gespeichert werden muß.

2.) Die andere Möglichkeit ist, Daten zu packen. Dies bedeutet, daß mehr Informationen in einem Byte gespeichert werden. Grundsätzlich besteht diese Möglichkeit immer, wenn Einschränkungen in der Darstellung hingenommen werden oder nur bestimmte Wertebereiche zugelassen sind.

Zwei solcher Fälle haben wir bereits in diesem Buch besprochen. Das Datum, bestehend aus 8 Zeichen, können wir dank des eingeschränkten Wertebereichs in nur 3 Zeichen unterbringen (s. Kapitel 3.7)

Logische Felder haben einen extrem eingeschränkten Wertebereich, sie kennen nur die Werte wahr (1) und falsch (0). Daher lassen sich in einem Byte 8 solcher Felder unterbringen (s. Kapitel 3.8)

Besonders gut lassen sich auch numerische Felder komprimieren. Hier spielt es keine Rolle, ob diese Felder

nur ganzzahlige Zahlen oder auch Dezimalbrüche enthalten dürfen.

Um eine Zahl zu speichern werden maximal 4 Bits benötigt :

0000 = 0
0001 = 1
0010 = 2
0011 = 3
0100 = 4
0101 = 5
0110 = 6
0111 = 7
1000 = 8
1001 = 9

Es bleiben noch 6 Kombinationen übrig, die als Code für Vorzeichen und Dezimalpunkt verwendet werden können, z.B.:

1010 = "-"
1011 = "."

Da ein Byte bekanntermaßen aus 8 Bits besteht folgt daraus, daß sich zwei Zahlen darin unterbringen lassen.

Das folgende Programm soll dies demonstrieren. Es verlangt die Eingabe einer zweistelligen Zahl und gibt den gepackten Code aus.

```
100 Z$=""
110 FOR I=1 TO 2
120 GET E$ : IF E$ "0" OR E$ "9" THEN 120
130 Z$ = Z$ + E$ : NEXT I
140 B = (ASC(LEFT$(Z$,1))-48)*16
150 B = B + ASC(RIGHT$(Z$,1))-48
160 B$ = CHR$(B) : PRINT B
```

In den Zeilen 110 bis 130 werden zwei Zahlen als String eingelesen und in Z$ zwischengespeichert.

In Zeile 140 und 150 werden diese dann umgerechnet und gepackt. Dann wird vom ASCII-Code der Zahlen, der zwischen 48 und 57 liegt, 48 subtrahiert. Man erhält eine Zahl zwischen 0 und 9. Um diese in die oberen 4 Bits zu transferieren, wird die Zahl mit 16 multipliziert.

Anschließend wird die zweite Zahl genauso behandelt, nur daß die Multiplikation nicht durchgeführt wird.

Damit ist in B der Code für die zwei Zahlen berechnet worden. Mit Hilfe des CHR$-Befehls wird die Zahl, die zwischen 0 und 255 liegen muß, in einen String von der Länge 1 umgewandelt. Damit haben wir zwei Zahlen in einem Byte gespeichert.

Nun der umgekehrte Weg, das Entpacken. Die Zahl steht wieder in B, die Zahlen werden in Klarschrift in der Variablen Z$ gespeichert.

Sie können dieses Programm einfach an das Vorherige anhängen. Es erscheinen dann nacheinander der Code für die zwei Zahlen und danach die Zahlen wieder in Klarschrift.

```
200 Z$=""
210 Z = B AND 240 : Z = Z + 48 : Z$ = CHR$(Z)
220 Z = B AND 15 : Z = Z/16 + 48 : Z$ = Z$ + CHR$(Z)
230 PRINT Z$
```

Sie sehen, der umgekehrte Weg ist einfacher. Zuerst werden die unteren vier Bits mit Hilfe des AND-Befehls ausgeblendet und dann die Rechnung aus dem vorherigen Programm rückwärts

ausgeführt. Dann werden die oberen 4 Bits ausgeblendet, wieder umgerechnet und in Z$ abgespeichert.

Natürlich lassen sich auch Buchstaben packen, allerdings nur, wenn Einschränkungen in der Darstellung in Kauf genommen werden. So kann z.B. nur Groß- oder nur Kleinschreibung erlaubt sein. Damit lassen sich 1 oder 2 Bits pro Buchstabe sparen. Dies mag auf den ersten Blick wenig erscheinen, summiert sich aber bei größeren Datenbeständen.

Sie sehen, es gibt vielfältige Möglichkeiten, Daten zu packen. Welche die geeignete ist, hängt von Ihrem speziellen Anwendungsfall ab. Sicher fallen Ihnen noch mehr Möglichkeiten ein.

KAPITEL 4 : PROGRAMMBEISPIELE

4.1 Einleitung

Zu einem Buch wie "64 für den Profi" gehört nicht nur eine Menge Theorie, sondern auch die Anwendung der einzelnen Kapitel soll Ihnen gezeigt werden. Wir haben zu den verschiedenen Beispielen, die Sie in diesem Buch gesehen haben Programme entworfen, die Ihnen die Umsetzung verschiedener Techniken in ein Programm verdeutlichen sollen. Da ist zum Beispiel eine parametrisierte Lagerverwaltung, die Ihnen zeigt, wie sich Parameter-Dateien in der Praxis einsetzen lassen. Außerdem zeigt Ihnen dieses Programm auch die Verwendung von relativen Dateien auf dem Commodore 64 zusammen mit der Floppy 1541. Oder, um noch andere Programme zu nennen, unser Literaturverzeichnis-Programm, in dem Sie eine "neue" Methode der Dateiverwaltung finden: Von uns QUISAM genannt (QUISAM steht hier für QUasi ISAM, also eine index-sequentielle Dateiverwaltung. Aber auch eine kleine Textverarbeitung finden Sie in diesem Kapitel - Sie können zwar mit diesem Programm keine Textbearbeitungssystem ersetzen, aber trotzdem ist auch dieses Programm, wie alle anderen auch, nicht nur als Beispielsprogramm gedacht sondern auch für den direkten Einsatz zu benutzen. Damit Sie als Programmierer keine Schwierigkeiten haben, diese Programme gegebenenfalls abzuändern, damit Sie auch Ihren speziellen Anforderungen genügen, haben wir uns bemüht, alle Programme sehr einfach und verständlich zu halten.

Haben Sie bitte Verständnis dafür, daß nicht jeder Anwender auf "seine Kosten" kommt - eine universelle Lösung für Ihre Probleme können wir hier selbstverständlich auch nicht anbieten. Das ist aber auch nicht das Ziel der nachfolgenden Beispielprogramme. Tippen Sie diese Programme nicht einfach ab, sondern steigen Sie tiefer in die Struktur der Programme ein und ändern diese entsprechend Ihren eigenen Vorstellungen. So können Sie nicht nur das in den ersten 3 Kapiteln gerlernte vertiefen, sondern auch gleichzeitig auf Basis dieser Programme komplexere, auf Ihre

Bedürfnisse zugeschnittene Anwendungen erstellen. Da aber, wie oben schon beschrieben, die Programme leicht verständlich gehalten wurden, dürfte jeder einigermaßen erfahrene Programmierer (und Profis wollen wir ja alle sein oder werden), die verschiedenen Programme abändern können. Außerdem werden zu den einzelnen Programme jedesmal genaue Erklärungen gegeben, die einem den Sinn des jeweiligen Programmteils erklären sollen.

Hinweis zum Aufbau der Programmerklärungen

Wir haben jedes Programm natürlich als komplettes Listing abgedruckt und uns bemüht alle Sonderzeichen durch die entsprechenden CHR$ zu ersetzen. Da Sie aber nicht nur einfach das Programm eintippen sollen, sondern auch etwas lernen wollen, so finden Sie vor jedem Programm 2 zusätzliche Teile.

Im ersten Teil erfahren Sie, wie Sie dieses Programm anwenden können. Dieser Teil ist zunächst einmal für alle die gedacht, die diese Programmsammlung direkt für eigene Zwecke gebrauchen möchten. Doch dieser Teil hat auch noch eine andere Bedeutung. So zeigt er allen Programmieren wie man ein Handbuch zu einem Programm schreiben kann. Und wie wichtig gerade das Handbuch für ein gutes Programm sein kann, haben wir ja schon in den ersten Kapiteln gelernt. Sie müssen sich natürlich nicht an die Art und Weise halten, wie wir diese Erklärungen geschrieben haben, sie sollen Ihnen nur als Hinweis dienen, wie man es machen kann.

Der zweite Teil schießlich befaßt sich mit dem Aufbau des Programms im einzelnen. Hier werden alle besonderen Programmierverfahren genau erklärt. Natürlich können wir nicht jedes PRINT und jedes INPUT genau erläutern, doch werden Sie über den Sinn eines jeden Programmteils genau aufgeklärt.

Und nun viel Spaß beim Eintippen der Programme, deren Lektüre und dem Lernen aus diesen Programmen.

4.2 Eine parameterisierte Lagerverwaltung

Teil 1:

Eines der vielen Probleme, die an Hand der EDV heutzutage viel leichter gelöst werden können als früher, ist das Problem der Lagerhaltung. Man möchte zu jedem Zeitpunkt erfahren können, wieviele Stücke man von einem Artikel noch auf Lager hat. Es muß hierbei berücksichtigt werden, daß durch den Verkauf jedesmal aus dem Lager abgebucht bzw. durch eine Lieferung die entsprechenden Artikel wieder in die Datei eingebucht werden sollen. Beide Aufgaben lassen sich mit diesem Programm erledigen.

Eine der Besonderheiten dieses Programms ist, wie der Name des Programms schon verkündet, die Verwendung einer sogenannten Parameterdatei. Mit dieser Datei lassen sich verschieden Parameter variabel in das Programm einbinden. Um das etwas zu verdeutlichen:

Stellen Sie sich vor, Sie haben verschiedene Mehrwertsteuer-Sätze. Üblicherweise beträgt der Mehrwertsteuersatz 7% bzw. 14%. Wenn Sie nun in Ihren Programmen mit diesen festen Werten arbeiten, so wird es schwer sein, dieses Wert bei einer Änderung in Ihrem ganzen Programm zu modifizieren. Wenn Sie sich nun noch vorstellen, daß Sie auch Programme verkaufen wollen, so läßt sich die Idee von festen Mehrwertsteuersätzen im Programm überhaupt nicht mehr realisieren, da verschiedene Unternehmen mit verschiedenen Mehrwertsteuersätzen arbeiten.

Noch weitaus aufwendiger wird es bei Firmenanschriften oder anderen Texten die von Unternehmen zu Unternehmen unterschiedliche Form haben. Sie können diese sogenannten Brief- oder Rechnungsköpfe zwar fest in das Programm einbauen - aber so arbeitet kein Profi - und das wollen wir ja werden.

Softwarehäuser benutzen bei Ihrer Programmierung sehr viele Parameterdateien in den unterschiedlichsten Formen. Es gibt zum

Ist dieser Vorgang beendet, meldet sich der Computer wieder mit seinem Hauptmenü.

```
*************************************
*                                   *
*   Parameterisierte Lagerverwaltung   *
*                                   *
*************************************

              HAUPTMENUE

        1) Eingeben der Artikeldatei

        2) Aendern der Artikeldatei

        3) Eingeben der Kassenzettel

        4) Lagerzugang

        5) Drucken der Artikelliste

        6) Ende des Programms


  -> Bitte waehlen Sie (1-6) :
```

Sie können nun durch einfaches Eintippen der entsprechenden Taste, aus diesen verschiedenen Menüpunkten (Menü bedeutet also tatsächlich soviel wie "Speiseplan") den Punkt auswählen, den Sie nun ausführen möchten.

Zunächst haben Sie die Möglichkeit Ihre Artikel in die Datei einzutragen. Dieses geschieht mit dem Menüpunkt 1 - "Eingeben der Artikeldatei". Wollen Sie also zum Beispiel ein Buch mit dem Titel "Wie werde ich Millionär" in Ihr Lager einbuchen, so wählen Sie zu Anfang den Punkt 1 aus dem Hauptmenü aus. Es erscheint

dann ein neuer Bildschirm, auf dem Sie einfach die entsprechende Artikelnummer eintragen können, danach die Bezeichnung und schließlich noch die Mehrwertsteuergruppe. Folgendes ist bei den Eingaben zu beachten: Die Eingabe der Artikelnummer darf maximal den Wert der unter "Anzahl der Artikel" bei der Parameterdefinition angegebenen Zahl erhalten. Als Minimum ist die Zahl "1" festgelegt.

Die Bezeichnung kann mit Ausnahme von "," und ":" aus allen Zeichen bestehen, darf aber höchstens 20 Zeichen lang sein.

Bei der Mehrwertsteuergruppe können Sie alle Zahlen zwischen 1 und 9 angeben, doch werden nur die Mehrwertsteuergruppen 1 und 2 mit den Prozentsätzen versehen, die Sie bei der Parameterdefinition angegeben haben. Alle anderen Sätze werden mit 0% behandelt.

Hier nun wieder ein Beispiel:

```
****************************************
*                                      *
*   Parameterisierte Lagerverwaltung   *
*                                      *
****************************************


          Aendern der Artikeldatei

1) Artikelnummer :? 125 <RETURN>

2) Bezeichnung   :? Videocassette <RETURN>

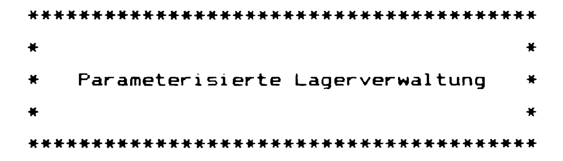
3) MWSt. Satz    :? 2 <RETURN>
```

So können Sie alle Ihre Artikel in die Lagerdatei eintragen. Wenn Sie keine weiteren Artikel mehr eintragen wollen, tippen Sie

aufleuchtet. Sollten Sie eine ganz neue Diskette verwenden, so wird das rote Lämpchen nun blinken.
Als nächstes stellt Ihnen der Computer die Frage "Neue Datendiskette anlegen (J/N) ?". An dieser Stelle müssen Sie dem Computer nun mitteilen, ob Sie eine neue Diskette verwenden wollen (oder eine Diskette die noch keine Parameterdatei enthält) oder eine Diskette die schon verschiedene Daten und die Parameterdatei enthält. Antworten Sie mit "J" oder "j" so wird die Diskette, die sich zu diesem Zeitpunkt in dem Laufwerk befindet, auf jeden fall neu formatiert.

Nach ein paar Sekunden meldet sich der Computer wieder und fordert Sie auf verschiedene Werte einzugeben. Wenn Sie eine Eingabe getätigt haben, drücken Sie einfach die RETURN-Taste und der Computer fährt fort.

Hier ein Beispiel wir Ihre Eingaben aussehen könnten:



```
****************************************
*                                      *
*   Parameterisierte Lagerverwaltung   *
*                                      *
****************************************
```

MWSt. Satz 1 : ?7 <RETURN>

MWSt. Satz 2 : ?14 <RETURN>

Anzahl der Artikel (max. 3000) : ?500 <RETURN>

Nach der letzten Eingabe kann es ein paar Minute dauern, bis sich der Computer wieder meldet. In dieser Zeit wird das Laufwerk dauernd laufen. Dies ist aber normal, da der Computer nun Ihre Diskette formatiert und anschließend die Parameter- und die relative Datei generiert.

Beispiel Dateien, die nur die verschiedenen Parameter enthalten, oder Dateien, bei den der erste Datensatz die Parameter enthält, die anderen Sätze dagegen die normalen Eintragungen, es gibt sequentielle oder indexsequentielle Parameterdateien, Dateien die bei Start des Programmes einmal eingelesen werden oder solche, die kontinuierlich auf die Parameter zugreifen und viele mehr.

Wir wollen uns an dieser Stelle mit der einfachsten und gebräuchlichsten Form beschäftigen.

Unsere Parameterdatei ist sequentiell gespeichert und wird zu Beginn des Programmes einmal eingelesen. Während der Arbeit wird dann nur noch mit den eingelesenen Variablen gearbeitet.

Um solch eine Parameterdatei zu erstellen, ist es bei unserem Programm notwendig eine neue Datendiskette zu erstellen. Wenn Sie das Programm mit RUN starten, werden Sie als erstes gefragt, ob das Floppylaufwerk auch wirklich eingeschaltet ist und ob die Geräteadresse des Laufwerks auch tatsächlich 8 ist. Sollte dieses nicht der Fall sein, so müssen Sie erst die erforderlichen Bedingungen erfüllen, also Floppy einschalten und/oder die Geräteadresse auf 8 stellen, bevor Sie mit dem Programm arbeiten können.

Antworten Sie auf die Frage mit "J" oder "j" so wird der Computer Sie dazu auffordern eine Datendiskette in das Laufwerk zu legen. Diese Datendiskette kann entwerder ganz neu sein oder auch schon verschiedene Daten enthalten. Sollte diese Diskette schon formatiert sein, aber die Parameterdatei noch nicht enthalten, so müssen Sie auch diese Diskette im nächsten Arbeitsgang formatieren. Aber denken Sie immer daran: WENN SIE EINE DISKETTE FORMATIEREN, GEHEN ALLE INFORMATIONEN, DIE SICH VORHER AUF DIESER DISKETTE BEFUNDEN HABEN, VERLOREN !!!

Nachdem Sie die Datendiskette eingelegt haben, drücken Sie auf die Meldung "Datendiskette einlegen" einfach eine beliebige Taste. Sie werden nun sehen, daß die rote Lampe an der Floppy

einfach "ENDE" und drücken die RETURN-Taste. Der Computer wird nun die Datei wieder schließen und Sie kehren ins Hauptmenü zurück.

Der zweite Punkt "Aendern der Artikeldatei" ähnelt sehr dem ersten. Sie haben hier die Möglichkeit bestehende Eintragungen der Artikeldatei zu ändern oder zu löschen. Dazu geben Sie einfach die Artikelnummer des betreffendes Artikels an und der Computer wird diesen, sofern er schon erfasst wurde, auf dem Bildschirm darstellen. Sie können nun die Bezeichnung bzw. die Mehrwertsteuergruppe des Artikels durch einfaches überschreiben bzw. Löschen der bisherigenen Eintragungen ändern und wieder wegspeichern. Wollen Sie ein bestimmtes Feld nicht ändern, so drücken Sie einfach die RETURN-Taste und der Computer wird die bisherige Eintragung übernehmnen.

Eine Besonderheit besteht in dem Sonderzeichen mit dem lustigen Namen "Klammeraffe". Dieses Zeichen finden Sie zwischen dem "P" und dem "*" auf der Tastatur des Commodore 64. Wenn Sie dieses Zeichen an die erste Stelle der Bezeichnung schreiben, so teilen Sie dem Computer dadurch mit, daß dieser Artikel gelöscht werden soll. Tatsächlich bleibt er aber in der Datei erhalten und kann durch erneutes Ändern des Klammeraffen in ein anderes Zeichen wieder "zum Leben erweckt werden".

Auch hier besteht wieder die Möglichkeit durch Eintragung von "ENDE" als Artikelnummer wieder ins Hauptmenü zurückzukehren.

Der dritte Punkt gibt Ihnen die Möglichkeit Kassenzettel einzugeben und somit dem Computer mitteilen, daß diese Artikel aus dem Lager abgebucht werden sollen. Gleichzeitig werden diese Eintragungen auf einem Drucker protokolliert. Wichtig ist es in diesem Zusammenhang darauf hinzuweisen, daß dieses Programm für einen Commodore 1525 oder Seikosha GP 100 VC Drucker geschrieben wurde. Falls Sie einen anderen Drucker besitzen, müssen Sie verschiedene Zeilen abändern. Aber hierauf kommen wir später noch zu sprechen.

Um einen Artikel aus dem Lager abzubuchen, geben Sie ganz einfach die entsprechende Artikelnummer ein und der Artikel wird auf dem Bildschirm dargestellt. Danach geben Sie die verkaufte Menge des Artikels ein. Als nächstes folgt der Einzel-Einkaufspreis und dann der Einzel-Verkaufspreis des entsprechenden Artikels. Der Computer berechnet sich nun automatisch die jeweiligen Gesamtpreise und druckt die Ergebnisse aus. Gleichzeitig bucht er die angegebene Anzahl aus dem Lager ab. So haben Sie immer eine genaue Kontrolle über die vorhandenen Artikel.

Da aber nicht nur Artikel abgebucht werden müssen, sondern auch eine Überwachung der Lieferungen stattfinden soll, gibt es im Hauptmenü noch einen weiteren Punkt. Sie finden diesen unter Punkt 4 - Lagerzugang.

Auch hier geben Sie einfach den Artikel an, bei dem Sie einen Lagerzugang verbuchen wollen. Auf dem Bildschirm erscheint dann die entsprechende Artikelbezeichnung, der bisherige Lagerbestand und die Frage nach dem Lagerzugang. Um dieses zu verdeutlichen hier ein Beispiel:

```
***************************************
*                                     *
*   Parameterisierte Lagerverwaltung  *
*                                     *
***************************************


              Lagerzugang

1) Artikelnummer :125

2) Bezeichnung   :Videocassetten

3) Lagerbestand  : 100

-> Lagerzugang   :?120
```

Durch diese Eintragung haben Sie nun einen neuen Lagerbestand von 220. Sie können nach dieser Eingabe entweder weitere Kassenzettel eingeben oder aber durch Eingabe von "ENDE" bei Artikelnummer wieder ins Hauptmenü zurückkehren.

Der nächste Punkt im Menü dient zur Ausgabe der gesamten Artikelliste auf einem Drucker. Sie haben so einen geneuen Überblick über Ihren momentanen Lagerbestand, die vergebenen Artikelnummern und den aktuellen Preisen. Alle diese Informationen finden Sie "schwarz auf weiß" auf Ihrem Drucker.

Hier ein Beispiel wie eine solche Artikelliste aussehen könnte:

DATUM : 30.11.83

GESCHÄFT : TESTLADEN, DORFSTR. 1

Artikelnummer	Artikelbezeichnung	Bestand	EK	VK
100	Leercassetten	50	5.5	7.75
105	Leertonbänder	25	12.5	19.75
125	Videocassetten	225	15	23.5

Der Letzte Punkt aus dem Hauptmenü gibt Ihnen die Möglichkeit das Programm abzuschließen und ins BASIC zurückzukehren. Die Datendiskette kann nun wieder aus dem Laufwerk entfernt werden und das Gerät kann abgeschaltet werden.

Teil 2 :

Die ersten Zeilen dienen zur Wahl der Rahmen-, Hintergrund- und Schriftfarbe. So geben

```
POKE 53280,14: REM RAHMENFARBE
POKE 53281,14: REM HINTERGRUND
PRINT CHR$(31): REM SCHRIFT
```

dem Bildschirm eine hellblaue Farbe und eine dunkelblaue Schrift. Durch eine identische Wahl von Rahmen- und Hintergrundfarbe vermeiden Sie außerdem noch die "sichtbare" Darstellung eines Bildschirmrahmens.

Nach diesem Teil wird wird als nächstes die Datendiskette gelesen. Zunächst wird durch OPEN 15,8,15,"I0" die Diskette initialisiert. Wenn gewünscht kann eine neue Diskette durch OPEN 15,8,15,"N:DATENDISKETTE,64" formatiert werden. Falls dies der Fall ist, wird auch gleich die relative Datei mit OPEN 1,8,3,"0:ARTDAT,L,"+CHR$(48) zur Bearbeitung vorbereitet. Hierbei bedeutet das "L" und der CHR$ hinter dem Programmnamen die Länge der zu Verarbeitenden Datei. In unserem Beispiel ist die Länge mit 48 Zeichen angegeben. Diese Länge setzt sich aus folgenden Feldern zusammen:

1. Artikelnummer = 4 Zeichen + <RETURN>
2. Bezeichnung = 20 Zeichen + <RETURN>
3. Lagerbestand = 4 Zeichen + <RETURN>
4. MWSt. Satz = 1 Zeichen + <RETURN>
5. Einzelpreis EK= 7 Zeichen + <RETURN>
6. Einzelpreis VK= 7 Zeichen

Summe = 48 Zeichen

Um die Werte später beim Einlesen und auch beim Schreiben in veschiedene Variablen speichern zu können, haben wir durch jeweils ein RETURN am Ende des Feldes dieses Feld von den anderen

getrennt. Das letze RETURN (hinter VK) setzt der Computer automatisch und darf nicht mitgerechnet werden.

Als nächstes wird die Parameterdatei erstellt. Dazu werden die jeweiligen Eingaben zunächst über ein normales INPUT eingelesen und dann in eine sequentielle Datei geschrieben. In dieser Datei stehen dann der Mehrwertsteuersatz 1, Satz 2 und die maximale Anzahl der Artikel. Dieser letzte Wert wird auch dazu benötigt, um die relative Datei zu erstellen.

```
OPEN 2,8,1,"0:PARAMETER"
PRINT#2,MS$(1)
PRINT#2,MS$(2)
PRINT#2,AN$
CLOSE 2
```

Dieses ist die Sequenz mit der die Parameterdatei auf Diskette geschrieben wird. Nun wird von der Anzahl der Artikel die Länge der relativen Datei berechnet. Dazu wird diese Anzahl in Low- und Highbyte zerlegt:

```
HB=INT(AR/256): LB=AR-HB*256
```

Eine Anzahl von 999 würde also zum Beispiel zu LB=231 und HB=3 führen. Mit diesen errechneten Werten können wir nun die relative Datei generieren. Dazu dienen folgende Befehle:

```
PRINT#15,"P"+CHR$(3)+CHR$(LB)+CHR$(HB)+CHR$(1)
PRINT#1,CHR$(255);
```

Die erste Anweisung schickt einen Befehl zum Kommandokanal, der dem System mitteilt, daß auf der Sekundäradresse 3 eine relative Datei bearbeitet wird, die (in unserem Beispiel) 999 Datensätze hat und daß die Verarbeitung vom ersten Zeichen innerhalb eines Datensatzes erfolgen soll. Durch die zweite Anweisung erstellt man die Datei wobei der letzte Datensatz als frei und generiert (=CHR$(255)) gekennzeichnet wird.

Danach beginnt die eigentliche Arbeit mit dem Programm.

Interessant ist hierbei wohl noch die besondere Art der Änderung eines Datensatzes. Dieser Datensatz wird einfach auf dem Bildschirm mit PRINT ausgegeben, durch einen CHR$(145) wird der Cursor über die entsprechende Eintragung gesetzt und mit einem INPUT läßt sich auf bequeme Art und Weise der Bildschirm-Editor ausnutzen.

TIP

Wenn Sie in Ihrem Programm auf einfache Art und Weise, sprich von BASIC aus den Cursor an eine bestimmte Stelle positionieren wollen, so sollten Sie folgende Variablen definieren:

```
HO$=CHR$(19): CU$=CHR$(17): CR$=CHR$(29)
FOR I=1 TO 7
  CU$=CU$+CU$: CR$=CR$+CR$
NEXT I
CU$=LEFT$(CU$,25): CR$=LEFT$(CR$,40)
```

Um nun zu einem bestimmten Punkt auf dem Bildschirm zu gelangen, genügt ein

```
PRINT HO$;LEFT$(CU$,VT);LEFT$(CR$,HT);
```

wobei VT den Wert der Zeile enthält und HT den Wert der Spalte.

Wenn Sie nun Daten speichern wollen, so müssen Sie wie gesagt daran denken, daß nach jedem Wort ein RETURN folgen muß, damit die Wörter auch richtig getrennt sind. Dazu dient folgende Zeile:

```
TE$=TE$(1)+CR$+TE$(2)+CR$+TE$(3)+CR$+TE$(4)+CR$+TE$(5)+CR$+TE$(6)
PRINT#8,TE$
```

Wir haben uns die Arbeit etwas erleichtert, da wir am Anfang des Programms CR$ direkt als CHR$(13) (=RETURN) definiert haben.

Um später wieder die Daten einlesen zu können, brauchen Sie nur die einzelnen Felder in der selben Reihenfolge wieder zu laden:

```
INPUT#8,TE$(1),TE$(2),TE$(3),TE$(4),TE$(5),TE$(6)
```

Die einzelnen Variablen enthalten dabei folgende Felder:

```
TE$(1)=Artikelnummer
TE$(2)=Bezeichnung
TE$(3)=Lagerbestand
TE$(4)=MWSt. Satz
TE$(5)=Einzelpreis EK
TE$(6)=Einzelpreis VK
```

Die Längen der einzelnen Felder erfahren Sie aus der Angabe der DATA Zeilen am Schluß des Programms. Wenn Ihnen verschiedene Felder zu kurz erscheinen, so können Sie diese leicht ändern. Denken Sie aber daran, dann auch die Längenangabe bei der Erstellung der relativen Datei (oben besprochen - Beispiel CHR$(48)) entsprechend zu ändern.
An dieser Stelle sollte auch noch auf die Verwendung des Fehlerkanals (vor den DATA-Zeilen) hingewiesen werden. Sicher ist Ihnen dessen Verwendung schon bekannt, aber trotzdem lohnt es sich den Aufbau einmal anzusehen. Die Fehler 0 und 50 (0=OK, 50=RECORD NOT PRESENT) können in unserem Programm übergangen werden, da die Abfrage auf die verschiedenen Datensätze automatisch durch das Programm gesteuert wird:

```
IF RN<1 OR RN>AN THEN .....
```

RN ist in diesem Fall der zu lesende/schreibende Datensatz und AN die maximale Anzahl der Datensätze. Durch diesen Befehl werden also alle Datensätze abgefangen, deren Nummer kleiner als 1 oder größer als die maximale Anzahl der möglichen Artikel ist.

```
ready.
 10000 rem parametrisierte lagerverwaltung
 10100 a=fre(0): clr
 10200 dim te$(9),td$(8),ti$(3),d1$(7),d2$(5),ms(3),mw(3)
 10300 ms(3)=1
 10400 cr$=chr$(13)
 10500 for i=1 to 8
 10600 read td$(i),td(i)
 10700 next
 10800 for i=1 to 3
 10900 read ti$(i),ti(i)
 11000 next
 11100 for i=1 to 7
 11200 read d1$(i)
 11300 next
 11400 for i=1 to 5
 11500 read d2$(i)
 11600 next
 11700 poke 53272,23: rem gross/klein
 11800 poke 53280,14: rem rahmenfarbe
 11900 poke 53281,14: rem hintergrund
 12000 print chr$(31);: rem schrift
 12100 gosub 41300
 12200 print "Diskettengeraet angeschlossen (J/N) ? ";
 12300 get ei$: if ei$="" then 12300
 12400 if ei$="J" or ei$="j" then 12600
 12500 goto 12300
 12600 print ei$: print
 12700 print "Datendiskette einlegen"
 12800 get ei$: if ei$="" then 12800
 12900 print: print
 13000 open 15,8,15,"i0"
 13100 close 15
 13200 print "Neue Datendiskette anlegen (J/N) ? ";
 13300 get ei$: if ei$="" then 13300
 13400 if ei$="N" or ei$="n" then print "N": goto 14500
 13500 if ei$="J" or ei$="j" then print "J": goto 13700
 13600 goto 13300
 13700 open 15,8,15,"n:datendiskette,64"
 13800 gosub 42100: gosub 41300
 13900 open 1,8,3,"0:artdat,l,"+chr$(48)
 14000 print#15,"p"+chr$(3)+chr$(lb)+chr$(hb)+chr$(1)
```

```
14100 print#1,chr$(255);
14200 rm=int(167132/48)
14300 close 1
14400 close 15
14500 open 2,8,0,"0:parameter"
14600 input#2,ms$(1): ms(1)=val(ms$(1))
14700 input#2,ms$(2): ms(2)=val(ms$(2))
14800 input#2,an$: an=val(an$)
14900 close 2
15000 gosub 41300
15100 print tab(15);"HAUPTMENUE": print: print
15200 print "      1) Eingeben der Artikeldatei": print
15300 print "      2) Aendern der Artikeldatei": print
15400 print "      3) Eingeben der Kassenzettel": print
15500 print "      4) Lagerzugang": print
15600 print "      5) Drucken der Artikelliste": print
15700 print "      6) Ende des Programms": print: print
15800 print "-> Bitte waehlen Sie (1-6) : ";
15900 get ei$: if ei$="" then 15900
16000 wahl=val(ei$)
16100 if wahl<1 or wahl>6 then 15900
16200 print ei$
16300 for i=1 to 1000: next
16400 on wahl goto 16600,19000,22600,32800,36600,40800
16500 end
16600 open 15,8,15
16700 open 8,8,8,"0:artdat"
16800 gosub 44900
16900 gosub 41300
17000 print tab(7);"Eingeben der Artikeldatei": print: print
17100 for i=1 to 3
17200 te$(i)=""
17300 print ti$(i);
17400 input te$(i)
17500 print
17600 if te$(1)="ENDE" or val(te$(1))<1 then 18700
17700 if len(te$(i))>ti(i) then 17200
17800 next
17900 for i=4 to 6
18000 te$(i)=""
18100 next
18200 rn=val(te$(1))
18300 if rn<1 or rn>an then 16900
18400 gosub 44000
```

```
18500 gosub 44600
18600 goto 16900
18700 close 8
18800 close 15
18900 goto 15000
19000 open 15,8,15
19100 open 8,8,8,"0:artdat"
19200 gosub 44900
19300 gosub 41300
19400 print tab(8);"Aendern der Artikeldatei": print: print
19500 te$(1)=""
19600 print ti$(1);
19700 input te$(1)
19800 print
19900 if te$(1)="ENDE" or val(te$(1))<1 then 22300
20000 if len(te$(1))>ti(1) then 19500
20100 rn=val(te$(1))
20200 if rn<1 or rn>an then 19300
20300 gosub 44000
20400 gosub 44400
20500 if val(te$(1))<>rn then 19300
20600 gosub 41300
20700 print tab(8);"Aendern der Artikeldatei": print: print
20800 print ti$(1);"  ";
20900 print te$(1)
21000 print
21100 for i=2 to 3
21200 print ti$(i);"? ";
21300 print te$(i)
21400 print chr$(145);
21500 print ti$(i);
21600 input te$(i)
21700 print
21800 if len(te$(i))>ti(i) then 21400
21900 next
22000 gosub 44000
22100 gosub 44600
22200 goto 19300
22300 close 8
22400 close 15
22500 goto 15000
22600 open 15,8,15
22700 open 8,8,8,"0:artdat"
22800 open 4,4,7
```

```
22900 gosub 44900
23000 gosub 41300
23100 gosub 43700
23200 print#4,"DATUM : ";da$
23300 print#4
23400 print#4,"GESCHAEFT : ";ge$
23500 print#4
23600 for i=1 to 79
23700 print#4,"-";
23800 next
23900 print#4
24000 print#4,chr$(16);"01";d1$(1);
24100 print#4,chr$(16);"10";d1$(2);
24200 print#4,chr$(16);"35";d1$(3);
24300 print#4,chr$(16);"40";d1$(4);
24400 print#4,chr$(16);"50";d1$(5);
24500 print#4,chr$(16);"60";d1$(5);
24600 print#4,chr$(16);"70";d1$(7)
24700 for i=1 to 79
24800 print#4,"-";
24900 next
25000 print#4
25100 gosub 41300
25200 print tab(7);"Eingeben der Kassenzettel": print: print
25300 te$(1)=""
25400 print td$(1);
25500 input te$(1)
25600 print
25700 if te$(1)="ENDE" or val(te$(1))<1 then 31100
25800 if len(te$(1))>td(1) then 25300
25900 rn=val(te$(1))
26000 if rn<1 or rn>an then 25100
26100 gosub 44000
26200 gosub 44400
26300 if val(te$(1))<>rn then 25100
26400 gosub 41300
26500 print tab(7);"Eingeben der Kassenzettel": print: print
26600 print td$(1);"  ";
26700 print te$(1)
26800 print
26900 print td$(2);"  ";
27000 print te$(2)
27100 print
27200 print td$(4);
```

```
27300 input te$(7)
27400 print
27500 if len(te$(7))>td(4) then 27200
27600 print td$(5);"? ";
27700 print te$(5)
27800 print chr$(145);
27900 print td$(5);
28000 input te$(5)
28100 print
28200 if len(te$(5))>td(5) then 27900
28300 print td$(7);"? ";
28400 print te$(6)
28500 print chr$(145);
28600 print td$(7);
28700 input te$(6)
28800 print
28900 if len(te$(6))>td(7) then 28600
29000 te$(4)=str$(val(te$(4))-val(te$(7)))
29100 if val(te$(4))<-999 then te$(4)="-999"
29200 te$(8)=str$(val(te$(5))*val(te$(7)))
29300 te$(9)=str$(val(te$(6))*val(te$(7)))
29400 ee=ee+val(te$(5)): ee$=str$(ee)
29500 ge=ge+val(te$(8)): ge$=str$(ge)
29600 ev=ev+val(te$(6)): ev$=str$(ev)
29700 gv=gv+val(te$(9)): gv$=str$(gv)
29800 gr=val(te$(3)): we=val(te$(9))
29900 if gr<>1 and gr<>2 then gr=3
30000 mw(gr)=mw(gr)+we*ms(gr)/100
30100 gosub 44000
30200 gosub 44600
30300 print#4,chr$(16);"01";te$(1);
30400 print#4,chr$(16);"10";te$(2);
30500 print#4,chr$(16);"35";te$(7);
30600 print#4,chr$(16);"40";te$(5);
30700 print#4,chr$(16);"50";te$(8);
30800 print#4,chr$(16);"60";te$(6);
30900 print#4,chr$(16);"70";te$(9)
31000 goto 25100
31100 for i=1 to 79
31200 print#4,"-";
31300 next
31400 print#4
31500 print#4,chr$(16);"01";"SUMMEN:";
31600 print#4,chr$(16);"40";ee$;: ee=0
```

```
31700 print#4,chr$(16);"50";g@$;: g@=0
31800 print#4,chr$(16);"60";ev$;: ev=0
31900 print#4,chr$(16);"70";gv$ : gv=0
32000 print#4
32100 print#4,chr$(16);"01";"MWSt. VK ";ms(1);"% = ";mw(1)
32200 print#4,chr$(16);"01";"MWSt. VK ";ms(2);"% = ";mw(2)
32300 print#4,chr$(16);"01";"MWSt. VK 0 % = ";mw(3)
32400 close 4
32500 close 8
32600 close 15
32700 goto 15000
32800 open 15,8,15
32900 open 8,8,8,"0:artdat"
33000 gosub 44900
33100 gosub 41300
33200 print tab(14);"Lagerzugang": print: print
33300 te$(1)=""
33400 print td$(1);
33500 input te$(1)
33600 print
33700 ifte$(1)="ENDE" or val(te$(1))<1 then 36300
33800 if len(te$(1))>td(1) then 33300
33900 rn=val(te$(1))
34000 if rn<1 or rn>an then 33100
34100 gosub 44000
34200 gosub 44400
34300 if val(te$(1))<>rn then 33100
34400 gosub 41300
34500 print tab(14);"Lagerzugang": print: print
34600 print td$(1);"  ";
34700 print te$(1)
34800 print
34900 print td$(2);"  ";
35000 print te$(2)
35100 print
35200 print td$(3);"  ";
35300 print te$(4)
35400 print
35500 print "-> Lagerzugang   :";
35600 input te$(7)
35700 if len(te$(7))>td(4) then 35500
35800 te$(4)=str$(val(te$(4))+val(te$(7)))
35900 if val(te$(4))>9999 then te$(4)="9999"
36000 gosub 44000
```

```
36100 gosub 44600
36200 goto 33100
36300 close 8
36400 close 15
36500 goto 15000
36600 open 15,8,15
36700 open 8,8,8,"0:artdat"
36800 open 4,4,7
36900 gosub 44900
37000 gosub 41300
37100 gosub 43700
37200 print#4,"DATUM : ";da$
37300 print#4
37400 print#4,"GESCHAEFT : ";ge$
37500 print#4
37600 for i=1 to 79
37700 print#4,"-";
37800 next
37900 print#4
38000 print#4,chr$(16);"01";d2$(1);
38100 print#4,chr$(16);"10";d2$(2);
38200 print#4,chr$(16);"35";d2$(3);
38300 print#4,chr$(16);"40";d2$(4);
38400 print#4,chr$(16);"60";d2$(5)
38500 for i=1 to 79
38600 print#4,"-";
38700 next
38800 print#4
38900 for rn=1 to an
39000 gosub 44000
39100 gosub 44400
39200 if val(te$(1))<>rn then 39900
39300 if left$(te$(2),1)="§" then 39900
39400 print#4,chr$(16);"01";te$(1);
39500 print#4,chr$(16);"10";te$(2);
39600 print#4,chr$(16);"35";te$(4);
39700 print#4,chr$(16);"40";te$(5);
39800 print#4,chr$(16);"60";te$(6)
39900 next
40000 for i=1 to 79
40100 print#4,"-";
40200 next
40300 print#4
40400 close 4
```

```
40500 close 8
40600 close 15
40700 goto 15000
40800 close 4
40900 close 8
41000 close 15
41100 print chr$(147)
41200 end
41300 print chr$(147);
41400 print "****************************************";
41500 print "*                                      *";
41600 print "*   Parametrisierte Lagerverwaltung    *";
41700 print "*                                      *";
41800 print "****************************************";
41900 print: print
42000 return
42100 gosub 41300
42200 input "MWSt. Satz 1 : ";ms$(1): ms(1)=val(ms$(1)): print
42300 if val(ms$(1))<1 or val(ms$(1))>50 then 42200
42400 input "MWSt. Satz 2 : ";ms$(2): ms(2)=val(ms$(2)): print
42500 if val(ms$(2))<1 or val(ms$(2))>50 then 42400
42600 input "Anzahl der Artikel (max. 3000) : ";an$: print
42700 if val(an$)<10 or val(an$)>3000 then 42600
42800 ar=val(an$)
42900 hb=int(ar/256)
43000 lb=ar-hb*256
43100 open 2,8,1,"0:parameter"
43200 print#2,ms$(1)
43300 print#2,ms$(2)
43400 print#2,an$
43500 close 2
43600 return
43700 input "Heutiges Datum (TT.MM.JJ) : ";da$: print
43800 input "Geschaeft : ";ge$: print
43900 return
44000 hb=int(rn/256): lb=rn-hb*256
44100 print#15,"p"+chr$(8)+chr$(lb)+chr$(hb)+chr$(1)
44200 gosub 44900
44300 return
44400 input#8,te$(1),te$(2),te$(3),te$(4),te$(5),te$(6)
44500 return
44600 te$=te$(1)+cr$+te$(2)+cr$+te$(3)+cr$+te$(4)+cr$+te$(5)+cr$+te$
44700 print#8,te$
44800 return
```

```
44900 input#15,f1$,f2$,f3$,f4$
45000 if val(f1$)=0 or val(f1$)=50 then return
45100 print "Fehler: ";f1$;" ";f2$;" ";f3$;" ";f4$
45200 close 8
45300 close 15
45400 for i=1 to 6000: next
45500 goto 12100
45600 data "1) Artikelnummer :",4
45700 data "2) Bezeichnung   :",20
45800 data "3) Lagerbestand  :",4
45900 data "4) Menge         :",3
46000 data "5) Einzelpreis EK:",7
46100 data "6) Gesamtpreis EK:",8
46200 data "7) Einzelpreis VK:",7
46300 data "8) Gesamtpreis VK:",8
46400 data "1) Artikelnummer :",4
46500 data "2) Bezeichnung   :",20
46600 data "3) MWSt. Satz    :",1
46700 data "Nummer"
46800 data "Bezeichnung"
46900 data "Menge"
47000 data "Einzel EK"
47100 data "Gesamt EK"
47200 data "Einzel VK"
47300 data "Gesamt VK"
47400 data "Nummer"
47500 data "Bezeichnung"
47600 data "Menge"
47700 data "Preis EK"
47800 data "Preis VK"

ready.
```

VARIABLENLISTE - PARAMETRISIERTE LAGERVERWALTUNG

```
A       - Dummy Variable
TE$( )  - Feldinhalte 1 - 9
TD$( )  - Titel Bezeichnung 1
TD  ( ) - Feldlängen 1
TI$( )  - Titel Bezeichnung 2
TI  ( ) - Feldlängen 2
D1$( )  - Drucker überschrift 1
D2$( )  - Drucker überschrift 2
MS  ( ) - Mehrwertsteuersätze
MW  ( ) - Mehrwertstuererbeträge
CR$     - Hilfsvariable (=RETURN)
I       - Variable für Schleifenzähler
EI$     - Variable zum Einlesen eines Zeichens
LB      - Low-Byte des Datensatzes bei relativer Dateiverwaltung
HB      - High-Byte des Datensatzes bei relativer Dateiverwaltung
RM      - Maximale Anzahl möglicher Datensätze
AN      - Maximale Anzahl tatsächlicher Datensätze
WAHL    - Gewählter Menüpunkt
RN      - Zu lesender / schreibender Datensatz
EE      - Gesamtbetrag Einzelpreis EK
GE      - Gesamtbetrag Gesamtpreis EK
EV      - Gesamtbetrag Einzelpreis VK
GV      - Gesamtbetrag Gesamtpreis VK
GR      - Gewählte Mehrwertsteuergruppe
WE      - Mehrwertsteuerbetrag
EE$     - s. EE
GE$     - s. GE
EV$     - s. EV
GV$     - s. GV
```

4.3 Ein universeller Reportgenerator für sequentielle Dateien

Teil 1:

Mit diesem Programm haben Sie die Möglichkeit Ihre sequentiellen Dateien auf dem Drucker auszugeben. Zusätzlich können Sie die relativen Dateien ausgeben, deren Datenfelder jeweils durch ein RETURN getrennt sind. Allerdings benötigen Sie dazu ein paar Informationen über den Aufbau der Datei. So müssen Sie zum Beispiel wissen, wieviele Felder innerhalb eines Datensatzes enthalten sind und wie die Titel der Felder lauten.

Die Bedienung des Programmes ist denkbar einfach. Sie geben einfach den Namen der Datei an die gedruckt werden soll und bestimmen danach die Feldanzahl pro Datensatz.

Der Computer fragt Sie danach nach den Titeln und Längen der Felder, die gedruckt werden sollen. Die Maximale Länge aller Felder zusammen darf 80 nicht übersteigen. Sie müssen jedoch eins bedenken: Wenn Sie als Titel eines Feldes POSTLEITZAHL eingeben und die Länge des Felds ist 4 Zeichen, so wird der Computer 12 Zeichen reservieren, da das Wort POSTLEITZAHL 12 Zeichen lang ist und ja ebenfalls gedrucket werden muß.

Nach diesen Eingaben wird der Computer alle Datensätze nacheinander von der Floppy lesen und auf dem Drucker in selber Form ausgeben. Nachdem der letzte Datensatz gedruckt wurde, haben Sie die Möglichkeit entweder eine weietere Datei auszudrucken oder aber das Programm zu beenden.

Teil 2:

Genau so einfach wie die Anleitung zu diesem Programm ist eigentlich auch das Programm selber. Interessant hierbei ist eigentlich tatsächlich wie einfach es ist relative Dateien sequentiell zu bearbeiten.

Genau wie unserem Programm zur parametrisierten Dateiverwaltung, so wird auch hier zunächst eine Datei zum Lesen geöffnet:

```
OPEN 8,8,8,DN$+",S,R"
```

wobei in DN$ der Name der Datei steht die auf dem Drucker ausgegeben werden soll. Danach folgt eine große Schleife, die alle Datensätze nacheinander in den Speicher liest, diejenigen Felder aussucht, die gedruckt werden sollen und dann diese Felder an den Druckerkanal ausgibt. Dieser Vorgang wird solange wiederholt bis der END OF FILE Merker erreicht ist. Danach wird die Datei und der Druckerkanal wieder geschlossen.

```
ready.
 10000 rem reportgenerator
 10100 a=fre(0): clr
 10200 dim ti$(99),fe$(99),te$(99)
 10300 poke 53272,23: rem gross/klein
 10400 poke 53280,14: rem rahmenfarbe
 10500 poke 53281,14: rem hintergrund
 10600 print chr$(31);: rem schrift
 10700 print chr$(147);
 10800 print "***          Reportgenerator          ***": print: print
 10900 print "-> Heutiges Datum (TT.MM.JJ) :";
 11000 input da$
 11100 print chr$(147);
 11200 print "***          Reportgenerator          ***": print: print
 11300 print "-> Name der Datei :";
 11400 input dn$: print
 11500 if dn$="" then 16800
 11600 open 15,8,15
 11700 open 8,8,0,"0:"+dn$+",s,r"
 11800 open 4,4,7
 11900 gosub 17300
 12000 print "-> Anzahl der Felder :";
 12100 input af$: print
 12200 if val(af$)<1 or val(af$)>99 then 12000
 12300 af=val(af$)
 12400 print "-> Anzahl Druckfelder :";
 12500 input ad$: print
 12600 if val(ad$)<1 or val(ad$)>af then 12400
 12700 ad=val(ad$)
 12800 for i=1 to ad
 12900 print "-> Titel #";i;" :";
 13000 input ti$(i)
 13100 if len(ti$(i))>20 then ti$(i)=left$(ti$(i),20)
 13200 print "-> Feldnummer :";
 13300 input fe$(i)
 13400 print
 13500 if val(fe$(i))<1 or val(fe$(i))>af then 12900
 13600 next
 13700 for i=1 to 79
 13800 print#4,"-";
 13900 next
```

VARIABLENLISTE - REPORTGENERATOR

A	- Dummy Variable
TI$( )	- Feldbezeichnung
FE$( )	- Feldnummer
TE$( )	- Feldinhalt
AF$	- Anzahl der Felder
AF	- s. AF$
AD$	- Anzahl Druckfelder
AD	- s. AD$
I	- Variable für Schleifenzähler
EOF	- Zeiger ob Ende der Datei erreicht

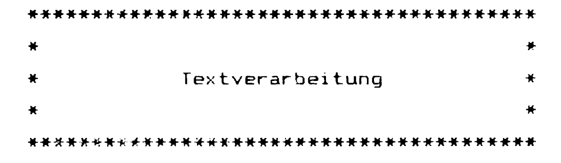
```
14000 print#4
14100 print#4,"DATUM : ";da$
14200 print#4
14300 print#4,"DATEINAME : ";dn$
14400 for i=1 to 79
14500 print#4,"-";
14600 next
14700 print#4
14800 for i=1 to af
14900 input#8,te$(i)
15000 next
15100 if st and 64 then eof=1
15200 for i=1 to ad
15300 print#4,chr$(16);"01";ti$(i);
15400 print#4,chr$(16);"22";":";
15500 print#4,chr$(16);"25";left$(te$(val(fe$(i))),54)
15600 next
15700 print#4
15800 if eof then 16000
15900 goto 14800
16000 for i=1 to 79
16100 print#4,"-";
16200 next
16300 print#4
16400 close 4
16500 close 8
16600 close 15
16700 goto 11100
16800 close 4
16900 close 8
17000 close 15
17100 print chr$(147)
17200 end
17300 input#15,f1$,f2$,f3$,f4$
17400 if val(f1$)=0 then return
17500 print "Fehler: ";f1$;" ";f2$;" ";f3$;" ";f4$
17600 close 4
17700 close 8
17800 close 15
17900 for i=1 to 6000: next
18000 goto 11100

ready.
```

4.4 Ein einfaches Textverarbeitungsprogramm

Zu jedem Computer gehört eine Textverarbeitung. Doch leider sind mit wenigen Ausnahmen die erhältlichen Textverarbeitungsprogramme für manche Anwendungsgebiete zu aufwendig und zu teuer. Wir können Ihnen an dieser Stelle ein kleines Textverarbeitungsprogramm vorstellen. Es ist zwar bei weitem nicht so leistungsfähig wie TEXTOMAT, dem DATA BECKER Textprogramm für den Commodore 64, aber es kann Ihnen bei der Eingabe von kleinen Texten sehr behilflich sein und Ihnen kleinere Schreibarbeiten abnehmen.

Nachdem Sie das Programm durch RUN gestartet haben, meldet sich folgendes Hauptmenü:



HAUPTMENUE

1) Texteingabe

2) Textaenderung

3) Textausgabe auf Bildschirm

4) Textausgabe auf Drucker

5) Texte speichern

6) Texte laden

5) Programmende

Wenn Sie nun die Taste "1" drücken so fragt der Computer

Ab Zeilennummer :?

Sie geben nun die Zeilennummer ein, bei der Sie anfangen wollen Text einzugeben. Haben Sie in die selben Zeilen schon einmal einen Text geschrieben, so wird dieser Text überschrieben. Alle anderen Zeilen werden von dieser Eingabe nicht berührt.

Sie können in einer Zeile maximal 79 Zeichen eingeben. Danach führt der Computer automatisch ein RETURN aus. Denken Sie also bei der Eingabe eines Textes daran, früh genug daran ein RETURN einzugeben. Dadurch wird die Zeile beendet. Auf dem Bildschirm erscheint dann die nächste Zeile und Sie können weiteren Text eingeben. Dieser Vorgang dauert solange, bis die Zeile 99 erreicht ist, dies ist die letzte mögliche Zeilennummer, oder wie Sie zu Anfang der Zeile die Funktionstaste F1 drücken. Sie kehren dann wieder ins Hauptmenü zurück.

Nun können Sie durch Drücken der Taste "2" den bestehenden Text korrigieren. Der Computer fragt Sie:

Welche Zeilennummer :?

Sie können nun die Zeilennummer eingeben, die Sie ändern wollen. Der Computer zeigt Zeilennummer und Zeile an und Sie können die Zeile neu eingeben. Nachdem Sie die RETURN-Taste gedrückt haben, fragt der Computer erneut nach einer Zeilennummer. Drücken Sie an dieser Stelle einfach die RETURN-Taste so gelangen Sie wieder ins Hauptmenü.

Wenn Sie mit dem Cursor nach rechts fahren, so werden Sie feststellen, daß sich die alte Zeile auf diese Art und Weise in die neue Zeile kopieren läßt. Drücken Sie die Pfeil-Taste nach links so werden die Zeichen die sich links vom Curso befinden wieder gelöscht (nur in der neuen Zeile !).

Drücken Sie als erstes Zeichen der Zeile einfach die RETURN-

Taste, so wird die alte Zeile vollständig in die neue Zeile übernommen. Ansonsten wird die neue Zeile bei einem RETURN abgeschlossen. Die neue Zeile befindet sich nun wieder an der richtigen Stelle im Text.

Nun können Sie den ganzen Text auf dem Bildschirm sichtbar machen. Dazu wählen Sie aus dem Hauptmenü den Punkt Ausgabe auf Bildschirm. Nach ein paar Sekunden erscheint der ganze Text beginnend mit Zeilennummer 1 auf dem Bildschirm. Durch Drücken der Leertaste können Sie den Bildschirm anhalten und wieder weiterlaufen lassen. Wenn alle Zeilen dargestellt wurden, meldet sich der Computer wieder mit

ENDE

Der Bildschirm bleibt aber noch sichtbar. Erst durch Drücken der Leertaste kehren Sie wieder in das Hauptmenü zurück.

Als weitere Möglichkeit können Sie die Ausgabe auch auf dem Drucker erfolgen lassen. Dazu wählen Sie Punkt "4" aus dem Hauptmenü. Der Computer fordert Sie auf den Drucker startklar zu machen und nachdem Sie die RETURN-Taste gedrückt haben beginnt der Druck. Nach Beendigung des Druckvorgangs kehrt das Programm wieder ins Hauptmenü zurück.

Die Punkte 5 und 6 erklären sich eigentlich selber. Sie können den ganzen Text auf Diskette abspeichern und wieder in den Speicher laden.

Um das Programm zu beenden, drücken Sie die Taste "6" im Hauptmenü.

Teil 2:

Bei einer Textverarbeitung ist es sehr wichtig zu jedem Zeitpunkt absoulte Kontrolle über die Tastatur zu haben. Diese Kontrolle kann mit der üblichen INPUT-Routine nicht gegeben werden. Das BASIC des Commodore 64 kennt dafür den GET-Befehl. Mit diesem Befehl können Sie zeichenweise einlesen und wissen so immer welche Taste gerade gedrückt wurde, ob diese Taste erlaubt ist und wie lang die Zeile im Moment gerade ist.

Mit einer einfachen Routine läßt sich die ganze Texteingabe kontrollieren:

```
100 GET EI$: IF EI$="" THEN 100
110 IF EI$>CHR$(31) AND EI$<CHR$(128) THEN 100
```

Diese Routine akzeptiert nur Zeichen die im sogenannten ASCII-Bereich liegt, das heißt, daß alle Sonder- und Steuerzeichen ignoriert werden.

Der GET-Befehl hilft uns aber auch noch an anderer Stelle. Wenn wir einen Text wieder einlesen wollen, so kann es passieren, daß wir nicht den ganzen Text einer Zeile eingelesen bekommen. Das liegt daran, daß die normale INPUT-Routine nur bis zum Komma einliest. Da aber innerhalb eines Textes sehr wohl Kommata vorkommen können, so müssen wir dafür sorgen, daß auch dieses Zeichen korrekt mit eingelesen wird.

Mit dieser Routine ist dies nun möglich:

```
100 OPEN 1,8,0,DN$;",S,R"
110 GET#1,EI$
120 TE$=TE$+EI$
130 IF EI$=CHR$(13) OR LEN(TE$)=80 THEN 150
140 GOTO 110
150 PRINT TE$
```

```
ready.
 10000 rem textverarbeitung
 10100 a=fre(0): clr
 10200 dim te$(100)
 10300 poke 53272,23: rem gross/klein
 10400 poke 53280,14: rem rahmenfarbe
 10500 poke 53281,14: rem hintergrund
 10600 print chr$(31);: rem schrift
 10700 print chr$(147);
 10800 print "****************************************";
 10900 print "*                                      *";
 11000 print "*           Textverarbeitung           *";
 11100 print "*                                      *";
 11200 print "****************************************";
 11300 print: print
 11400 print tab(15);"HAUPTMENUE": print: print
 11500 print "     1) Texteingabe ueber Tastatur": print
 11600 print "     2) Textaenderung": print
 11700 print "     3) Textausgabe auf Bildschirm": print
 11800 print "     4) Textausgabe auf Drucker": print
 11900 print "     5) Text speichern": print
 12000 print "     6) Text laden": print
 12100 print "     7) Programm Beenden": print
 12200 print "-> Bitte waehlen Sie (1-7) : ";
 12300 get ei$: ifei$="" then 12300
 12400 wahl=val(ei$)
 12500 if wahl<1 or wahl>7 then 12300
 12600 on wahl goto 12800,16000,19600,21300,22200,23200,24600
 12700 end
 12800 print chr$(147);
 12900 print "-> Ab Zeilennummer :";
 13000 input ze$
 13100 print
 13200 if val(ze$)<1 or val(ze$)>99 then 12900
 13300 ze=int(val(ze$))
 13400 print
 13500 print ze;": ";
 13600 az=0
 13700 get ei$
 13800 print chr$(185);chr$(157);
 13900 if ei$="" then 13700
 14000 if ei$<>chr$(20) then 14600
```

```
4100 if len(te$(ze))<1 then 13700
4200 print " ";chr$(157);
4300 print chr$(157);" ";chr$(157);
4400 te$(ze)=left$(te$(ze),len(te$(ze))-1)
4500 goto 13700
4600 if ei$<>chr$(133) then 14900
4700 if len(te$(ze))<1 then 15700
4800 goto 13700
4900 if az=0 then te$(ze)="": az=1
5000 print " ";chr$(157);ei$;
5100 if ei$=chr$(13) then ze=ze+1: goto 15400
5200 te$(ze)=te$(ze)+ei$
5300 if len(te$(ze))=79 then te$(ze)=te$(ze)+chr$(13): ze=ze+1
5400 if ze=100 then 15700
5500 if ei$=chr$(13) then 13500
5600 goto 13700
5700 print " ";chr$(157)
5800 te$(ze)="!!"
5900 goto 10700
6000 print chr$(147);
6100 print "-> Welche Zeilennummer :";
6200 input ze$
6300 print
6400 if val(ze$)<1 or val(ze$)>99 then 16100
6500 ze=int(val(ze$))
6600 print
6700 print ze;": ";
6800 print te$(ze)
6900 print
7000 print ze;": ";
7100 nt$=""
7200 get ei$
7300 print chr$(185);chr$(157);
7400 if ei$="" then 17200
7500 if ei$<>chr$(20) then 18100
7600 if len(nt$)<1 then 17200
7700 print " ";chr$(157);
7800 print chr$(157); " ";chr$(157);
7900 nt$=left$(nt$,len(nt$)-1)
300 goto 17200
100 if ei$<>chr$(29) then 18400
200 iflen(nt$)<=len(te$(ze)) then ei$=mid$(te$(ze),len(nt$)+1,1):goto 18800
00 goto 17200
400 if ei$<>chr$(133) then 18800
```

```
18500 if len(nt$)<1 then 10700
18600 goto 17200
18700 if az=0 then nt$="": az=1
18800 print " ";chr$(157);ei$;
18900 if ei$=chr$(13) then 19300
19000 nt$=nt$+ei$
19100 if len(nt$)=79 then nt$=nt$+chr$(13)
19200 goto 17200
19300 if len(nt$)<1 then nt$=te$(ze): print nt$
19400 te$(ze)=nt$
19500 goto 10700
19600 print chr$(147);
19700 ze=1
19800 if left$(te$(ze),2)="!!" then 20400
19900 print ze;": ";
20000 print te$(ze)
20100 gosub 20800
20200 ze=ze+1: ifze=100 then 20400
20300 goto 19800
20400 print
20500 print "** ENDE **";
20600 get ei$: if ei$="" then 20600
20700 goto 10700
20800 get ei$
20900 if ei$<>chr$(32) then return
21000 get ei$
21100 if ei$<>chr$(32) then 21000
21200 return
21300 open 4,4,2
21400 ze=1
21500 if left$(te$(ze),2)="!!" then 22000
21600 print#4,te$(ze)
21700 ze=ze+1
21800 if ze=100 then 22000
21900 goto 21500
22000 close 4
22100 goto 10700
22200 print chr$(147);
22300 print "-> Textname :";
22400 input dn$
22500 open 1,8,2,"§:"+dn$+",s,w"
22600 for ze=1 to 100
22700 if te$(ze)="" then te$(ze)=chr$(13)
22800 print#1,te$(ze)
```

```
22900 next
23000 close 1
23100 goto 10700
23200 print chr$(147);
23300 print "-> Textname :";
23400 input dn$
23500 open 1,8,0,dn$+",s,r"
23600 for ze=1 to 100
23700 te$=""
23800 get#1,ei$
23900 te$=te$+ei$
24000 if ei$=chr$(13) or ei$="" then 24200
24100 goto 23800
24200 te$(ze)=te$
24300 next
24400 close 1
24500 goto 10700
24600 print chr$(147)
24700 end

ready.
```

VARIABLENLISTE - TEXTVERARBEITUNG

A	- Dummy Variable
TE$( )	- Textzeilen 1 - 99
WAHL	- Gewählter Menüpunkt
I	- Variable für Schleifenzähler
ZE$	- Zeilennummer
ZE	- s. ZE$
AZ	- Hilfsvaraible - 0 bis erstes Zeichen eingegeben
EI$	- Variable zum Einlesen eines Zeichens
NT$	- Geänderte Zeile
DN$	- Dateiname

4.5 QUISAM konkret - eine Literaturstellenverwaltung

Nachdem wir Ihnen in Kapitel 2 alles beigebracht haben, was Sie für eine gute Dateiverwaltung wissen müssen und Ihnen eine Reihe von exzellenten Unterprogrammen an die Hand gegeben haben, wollen wir das Ganze nun einmal praktisch nutzen.

Dazu schreiben wir ein Programm, das Sie nicht nur gut gebrauchen können, sondern das auch die Möglichkeit von QUISAM, beliebig lange Datensätze zu verwalten, voll ausnutzt.

Wenn Sie eifriger Leser von Computerzeitschriften sind, kennen Sie sicher das Problem : Sie lesen einen Artikel zu einem bestimmten Thema und erinnern sich schwach "Moment mal, da gab es doch schon einmal etwas zu diesem Thema, wie war das noch" und noch viel schlimmer

WO STAND DAS NOCH GLEICH ???

Diesem Problem werden wir nun ein für alle mal auf den Leib rücken. Wir wollen eine Literaturstellenverwaltung schreiben, die es uns ermöglicht, Artikel anhand von Themenoberbegriffen zu suchen und uns angibt, wann und in welcher Zeitschrift ein Artikel zu diesem Thema erschienen ist.

Auf den ersten Blick werden Sie vielleicht denken, daß dies nichts besonderes ist. Wenn Sie aber einmal genau überlegen, was wir tun wollen, erkennen Sie sicher sehr schnell, wo der Haken bei einer Literaturstellenverwaltung liegt.

Eine normale relative Datei verlangt eine konstante Datensatzlänge, damit die Verwaltung der Diskette möglich ist. Aber woher sollen wir die nehmen? Wir haben zwar den Themenoberbegriff, für den wir eine bestimmte maximale Länge

definieren können, aber an wieviel Stellen ein Artikel zu diesem Thema erscheint können wir nicht wissen. Mit jeder neuen Zeitschrift kann diese Anzahl wachsen. Natürlich könnten wir eine maximale Anzahl von vielleicht 20 Einträgen vorsehen.
Damit hätten wir uns aber 2 Nachteile eingehandelt. Zuerst einmal ist bei 20 Einträgen Schluß, was tun wir dann ???
Der zweite Nachteil ist die enorme Platzverschwendung auf der Diskette, die sowieso keine übermäßig große Kapazität besitzt. Denn schließlich wird zu einem Thema mehr, zu einem anderen weniger geschrieben. Warum also für alle Themen gleich viel Platz vorsehen???
All dies ruft nach einer Dateistruktur mit dynamisch wachsender Datensatzlänge, eben nach QUISAM.
Die Einrichtung der Bemerkungsdatei erlaubt es, beliebig viele Stellen zu einem Thema einzutragen, und es wird immer nur soviel Platz reserviert wie tatsächtlich benötigt wird.

Und nun konkret. Dies soll das Programm können :

- Datei einrichten
- Thema eingeben
- Stelle eingeben
- Thema suchen
- Thema löschen
- Thema drucken

Thema drucken soll dabei bedeuten, daß zu einem Thema alle Stellen, an denen ein Artikel dazu erschienen ist, ausgedruckt werden.

Was wollen wir speichern ?

Wir brauchen genau zwei Eingabefelder :

- Thema
- Stelle

Thema enthält dabei den Themenoberbegriff, Stelle enthält Name der Zeitschrift, Erscheinungsdatum in Monat und Jahr (z.B. 7/82) und Seitenangabe.

Wie Sie nun vor allem die Themenoberbegriffe wählen bleibt Ihnen überlassen. Da sind Ihrer Fantasie keine Grenzen gesetzt.

Unser Stammdatensatz besteht also nur aus einem einzigen Feld, dem Thema. Dieses Feld ist gleichzeitig Schlüssel und kompletter Datensatzinhalt.
Alle Stellen werden in die Bemerkungsdatei eingetragen, die im Laufe der Zeit vom Umfang daher erheblich größer werden dürfte als die Stammdatei. Doch ist diese Datei nur genau so groß, wie Sie auch Stellen eingetragen haben. Im Gegensatz zu herkömmlichen Dateiverwaltungssystemen wird also nur der tatsächtlich benötigte Platz reserviert und Sie können beliebig viele Stellen zu einem Thema angeben.
Auf den folgenden Seiten finden Sie das Listing zu dem Programm. Sie werden erstaunt feststellen, wie kurz es ist und trotzdem alle Eigenschaften besitzt, die von einem solchen Programm erwartet werden. Zu dem Programm gehören aber auch noch eine Reihe von Unterprogrammen aus dem 2. Kapitel, die wir hier aus Platzgründen nicht noch einmal mit ausgedruckt haben. Wenn Sie das Kapitel 2 nach Anleitung durchgearbeitet haben, so werden Sie alle die benötigten Routinen schon eingetippt und gespeichert haben. Diese können Sie dann benutzen. Wenn nicht, so müssen Sie dies nun nachholen. Zuerst einmal eine Liste aller benötigten Unterprogramme mit Zeilennummern :

48000	formatierte Ausgabe
49000	Dateneingabe
50000	Defaultwerte eintragen
51000	Maske aufbauen
53000	Bemerkungseintrag lesen
53200	Folgebemerkung lesen
54000	Datensatz lesen

55000	Datensatz löschen
56000	Bemerkung eintragen
56600	Folgesatz eintragen
57000	Datensatz schreiben
58000	Datei öffnen
59000	Datei einrichten

Auf den folgenden Seiten nun das Listing, an das Sie die vorstehenden Unterprogramme wie gesagt noch anhängen müssen.

```
1000 REM
1010 REM ***  PROGRAMMSTART  ***
1020 REM
1025 GOSUB 1500 : REM  PROGRAMM INITIALISIEREN
1030 RESTORE
1040 FOR MP=1 TO 8 : REM HAUPTMENUE LESEN
1050 :READ M%(MP) : READ M$(MP) : READ MF%(MP) : READ MT%(MP)
1060 NEXT MP
1070 FOR MP=1 TO 5 : REM KOPFZEILEN LESEN
1080 :READ KO$(MP)
1090 NEXT MP
1100 READ TH$ : READ S$
1200 REM HAUPTMENUE SCHREIBEN
1210 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
1220 FOR MP=1 TO 8
1230 :F%=M%(MP) : F$=M$(MP) : FL%=MF%(MP) : FF%=MT%(MP)
1240 :GOSUB 51000 : REM MASKE DRUCKEN
1250 NEXT MP
1260 REM MENUEPUNKT WAEHLEN
1270 F%=17 : GOSUB 49000
1280 IF F$<"1" OR F$>"6" THEN 1270 : REM UNGUELTIGE EINGABE
1290 MP=VAL(F$) : ON MP GOSUB 3000,4000,5000,6000,7000,8000
1300 GOTO 1210 : WIEDER VON VORNE
1500 REM PROGRAMM INITIALISIEREN
1510 PRINT CHR$(147)
1520 F%=3 : F$="LITERATURSTELLENVERZEICHNIS" : FL%=0 : FF%=0 : GOSUB 51000
1530 F%=6 : F$="NAME DER DATEI : " : FL%=8 : FF%=0 : GOSUB 51000
1540 GOSUB49000:QN$=F$ :PRINT: REM NAMEN DER DATEI EINGEBEN
1560 DN$=QN$ + ".PR" : OPEN 15,8,15
1570 GOSUB 63500 : CLOSE 15 : REM IST DATEI VORHANDEN
1580 IF F1$="00" THEN GOSUB 58000 : RETURN : REM DATEI IST VORHANDEN
1590 GOSUB 59060 : RETURN : REM DATEI EINRICHTEN
3000 REM
3010 REM ***  THEMA EINGEBEN  ***
3020 REM
3030 GOSUB 40000 : REM MASKE AUFBAUEN
3040 F%=5 : F$=TH$ : FL%=25 : FF%=0 : GOSUB 51000 : GOSUB 49000 : QD$=F$
3045 IF F$=CHR$(137)THEN 3080
3050 F%=7 : F$=S$ : GOSUB 51000 : GOSUB 49000 : QX$=F$ : REM STELLE EINGEBEN
```

```
3060 IF F$=CHR$(137)THEN 3080
3065 GOSUB 57000 : REM THEMA AUF DISK SCHREIBEN
3070 GOSUB 56000 : REM STELLE SCHREIBEN
3075 IFVAL(F1$)<>0 THEN 41000 : REM FEHLER
3080 RETURN
4000 REM
4010 REM  ***  STELLE EINGEBEN  ***
4020 REM
4030 GOSUB 40000 : REM KOPF DRUCKEN
4040 F%=5 : F$=TH$ : FL%=25 : FF%=0 : GOSUB 51000
4050 F$=QD$ : GOSUB 50000
4060 F%=7 : F$=S$ : FL%=25 : GOSUB 51000 : GOSUB 49000 : REM STELLE EINGEBEN
4070 IF F$=CHR$(137) THEN 4100 : REM ABBRECHEN
4080 QX$=F$ :IFQT$=".EX"THENGOSUB56600:GOTO4090
4085 GOSUB 56000
4090 IF F1$<>"00"THEN 41000
4100 RETURN
5000 REM
5010 REM  *** THEMA SUCHEN  ***
5020 REM
5030 GOSUB 40000 : REM KOPF DRUCKEN
5040 F%=5 : F$=TH$ : FL%=25 : FF%=0 : GOSUB 51000
5050 F%=7 : F$=S$ : GOSUB 51000
5060 F%=5 : GOSUB 49000 : REM THEMA EINGEBEN
5070 IF F$=CHR$(137) THEN 5440 : REM ABBRUCH
5080 QK$=F$ : GOSUB 54000 : REM THEMA SUCHEN
5090 IF QF%>0 THEN 5400 : REM NICHT GEFUNDEN
5100 F$=QD$ : GOSUB 50000 : REM IN MASKE EINTRAGEN
5110 GOSUB 53100 : REM ERSTE STELLE LESEN
5120 IF QF%>0 THEN 5400 : REM NICHT GEFUNDEN
5130 F$=QX$ : F%=7 : GOSUB 50000 : REM IN MASKE EINTRAGEN
5140 F%=24 : F$="F7 - NAECHSTE STELLE / F2 - ABBRUCH" : FL%=0 : FF%=0
5150 GOSUB 51000
5160 GET E$ : IF E$=CHR$(137) THEN 5440 : REM ABBRUCH
5170 IF E$<>CHR$(136) THEN 5160
5180 REM NAECHSTE STELLE LESEN
5190 GOSUB 53200
5200 IF QF%>0 THEN 5400 : REM NICHT GEFUNDEN
5210 GOTO 5130
```

```
5400 REM NICHT GEFUNDEN
5410 F%=24 : F$="                                        " : FL%=0 : GOSUB 51000
5420 F$="DATENSATZ NICHT GEFUNDEN" : GOSUB 51000
5430 GET E$ : IF E$="" OR E$<>CHR$(13) THEN 5430
5440 RETURN
6000 REM
6010 REM  ***  THEMA LOESCHEN  ***
6020 REM
6030 GOSUB 40000 : REM KOPF DRUCKEN
6040 F%=5 : F$=TH$ : FL%=25 : FF%=0 : GOSUB 51000
6050 F$=QD$ : GOSUB 50000 : REM AKTUELLES THEMA AUF SCHIRM
6060 F%=7 : F$="WIRKLICH LOESCHEN (J/N)" : FL%=1 : FF%=1 : GOSUB 51000
6070 GOSUB 49000 : REM EINGABE
6080 IF F$="N" THEN 6110 : REM ABBRUCH
6090 IF F$<>"J"THEN 6070
6100 GOSUB 55000
6110 RETURN
7000 REM
7010 REM  ***  THEMA DRUCKEN  ***
7020 REM
7030 GOSUB 40000
7040 F%=5 : F$=TH$ : FL%=25 : FF%=0 : GOSUB 51000
7050 F$=QD$ : GOSUB 50000 : REM AKTUELLES THEMA AUF SCHIRM
7060 F%=7 : F$="DRUCKER FERTIG (J/N)" : FL%=1 : FF%=1 : GOSUB 51000
7070 GOSUB 49000
7080 IF F$="N" THEN 7300 : REM ABBRUCH
7090 IF F$<>"J" THEN 7070
7100 GOSUB 53100 : REM ERSTE STELLE LESEN
7110 IF QF%>0 THEN 7300 : ENDE DES DATENSATZES
7120 OPEN 1,4
7130 FI$=QD$ : FO$="AAAAAAAAAAAAAAAAAAAAAAAAA   AAAAAAAAAAAAAAAAAAAAAAAAA"
7140 GOSUB 48000 : REM AUSGABE FORMATIEREN
7150 FI$=QX$ : GOSUB 48000
7160 PRINT#1,FP$
7170 FO$="                            AAAAAAAAAAAAAAAAAAAAAAAAA"
7175 REM FORMAT FUER RESTLICHE STELLEN
7190 GOSUB 53200 : REM NAECHSTE STELLE LESEN
7200 IF QF%>0 THEN 7300 : REM ENDE DES DATENSATZES
7210 FI$=QX$ :  GOSUB 48000 : REM FORMATIEREN
```

```
7220 PRINT#1,FP$ : GOTO 7190
7300 CLOSE1 : RETURN
8000 REM
8010 REM PROGRAMM BEENDEN
8020 REM
8030 GOSUB 58300
8040 END
10000 REM
10010 REM STRINGS
10020 REM
10030 DATA 0,"HAUPTMENUE",0,0
10050 DATA 5,"1 : THEMA EINGEBEN",0,0
10060 DATA 7,"2 : STELLE EINGEBEN",0,0
10070 DATA 9,"3 : THEMA SUCHEN",0,0
10080 DATA 11,"4 : THEMA LOESCHEN",0,0
10090 DATA 13,"5 : THEMA DRUCKEN",0,0
10095 DATA 15,"6 : PROGRAMM BEENDEN",0,0
10100 DATA 17,"BITTE WAEHLEN SIE",1,1
10200 DATA "THEMA EINGEBEN"
10210 DATA "STELLE EINGEBEN"
10220 DATA "THEMA SUCHEN"
10230 DATA "THEMA LOESCHEN"
10240 DATA "THEMA DRUCKEN"
10300 DATA "THEMA  : " , "STELLE : "
40000 REM
40010 REM  ***  KOPFDRUCK  ***
40020 REM
40030 PRINT CHR$(147) : REM BILDSCHIRM LOESCHEN
40040 F%=0 : F$=KO$(MP) : FL%=0 : FF%=0 : GOSUB 51000 : REM KOPFZEILE DRUCKE
40050 RETURN
41000 REM
41010 REM  ***  FEHLERBEHANDLUNG  ***
41020 REM
41030 F%=24 : F$="FEHLER AUF DER DISKETTE" : FL%=0 : FF%=0 : GOSUB 51000
41040 GET E$ : IF E$="" OR E$<>CHR$(13) THEN 41040
41050 RETURN
```

Sie sehen schon anhand des Listings, daß bei Benutzung der Unterprogramme fast nichts mehr zu tun ist. Lediglich das Hauptmenue und die Eingaben sind noch aufgebaut worden. Ansonsten besteht das Programm fast nur aus Unterprogrammaufrufen. Dies ist auch bereits an der extrem kurzen Variablenliste zu sehen, die nur die Variablen enthält, die außerhalb der Unterprogramme benutzt werden.

MP angewählte Nummer des Menuepunktes
TH$ Konstante "Thema :"
S$ Konstante "Stelle :"
M% Zeilennummern der einzelnen Menuepunkte
M$ Menuepunkte
MF% Länge der Eingabefelder (0)
MT% Typ der Eingabefelder (0)
KO$ Kopfzeile der einzelnen Eingabemasken

Das Programm selber gliedert sich in vier große Teile :

Zeile 1000 - 2000 : Aufbau des Hauptmenues und Auswahl der einzelnen Unterprogramme

Zeile 2000 - 8000 : die einzelnen Unterprogramme

Zeile 10000 - 20000 : benötigte Konstanten

Zeile 40000 - 60000 : Unterprogramme

Die einzelnen Programmteile erklären sich fast von selbst. Trotzdem eine kurze Erläuterung :

1000 - 1200 : Initialisierung des Programms. Eingabe des Dateinamens und Belegung der konstanten Stringvariablen

1200 - 1300 : Hauptprogramm; druckt Hauptmenue aus und ermöglicht Auswahl der einzelnen Programmteile

1500 - 1590 : Eingabe des Dateinamens, wird diese Datei auf der Diskette nicht gefunden, so wird eine neue Datei eingerichtet.

3000 - 4000 : Thema eingeben, d.h. Datensatz eingeben und speichern

3030 : Kopfzeile drucken

3040 - 3050 : Eingabemaske aufbauen und Thema und Stelle einlesen

3060 : Thema speichern

3065 : Wenn keine Stelle eingeben, dann speichern überspringen

3070 : Stelle auf Diskette in Bemerkungsdatei schreiben

3075 : Wenn Fehler dann Fehlermeldung ausgeben

3080 : Zurück ins Hauptmenue

4000 - 4100 : Stelle zu vorhandenem Thema eingeben

4030 : Kopfzeile drucken

4040 - 4060 : Eingabemaske drucken und Stelle einlesen

4070 : Bei Leereingabe Abbruch

4080 : Stelle in Bemerkungsdatei speichern wenn vorher bereits auf diese Datei zugegriffen wurde.

4085 : Stelle in Bemerkungsdatei eintragen wenn

vorher noch nicht auf diese Datei zugegriffen wurde.

4090 : Wenn Fehler dann Fehlermeldung ausgeben

4100 : Zurück ins Hauptmenue

5000 - 5430 : Thema auf Diskette suchen und mit allen Stellen anzeigen

5030 : Kopfzeile drucken

5040 - 5060 : Eingabemaske drucken und Suchbegriff eingeben

5070 : Bei Leereingabe Abbruch

5080 : Thema auf Diskette suchen

5090 : Wenn nicht gefunden, dann entsprechende Meldung ausgeben

5100 : Thema in Maske eintragen

5110 - 5130 : Erste Stelle suchen und in Maske eintragen

5140 - 5170 : Soll weitergesucht werden ?

5180 - 5210 : Nächste Stelle suchen

5400 - 5430 : Fehlermeldung drucken und zurück ins Hauptmenue

6000 - 6110 : Thema löschen

6030 - 6090 : Sicherheitsabfrage

6100 : Thema löschen

7000 - 7300 : Thema drucken

7040 - 7090 : Sicherheitsabfrage

7100 : Erste Stelle lesen

7120 : Druckerkanal öffnen

7130 - 7140 : Formatierung der Ausgabe; die Formatanweisung in FO$ können Sie natürlich nach eigenem Belieben ändern.

7160 : Ausgabe von Thema und erster Stelle

7170 : Neues Format für restliche Stellen

7190 - 7220 : Naechste Stelle lesen und ausgeben

7300 : Druckerkanal schließen und zurück ins Hauptmenue

10000-10100 : Strings für das Hauptmenue, alle DATA-Zeilen sind nach dem Muster Zeilennummer, String, Länge des Eingabefeldes, Typ des Eingabefeldes aufgebaut.

10200-10240 : Kopfzeilen der einzelnen Unterprogramme

Kurz einiges zur Bedienung des Programms :

Beim Start verlangt das Programm den Namen der Datei. Findet es eine Datei diesen Namens nicht auf der Diskette so richtet es eine neue Datei ein. Geben Sie dazu die Datensatzlänge 25 ein, die Länge des Bemerkungssatzes ebenfalls mit 25, die Schlüsselposition mit 1 und die Schlüssellänge auch mit 25. Diese Werte sind natürlich beliebig und können von Ihnen verändert werden. Allerdings müssen Sie dann auch die Druckausgabe entsprechend ändern.

Vom Haupmenue aus können Sie die einzelnen Programmteile einfach durch Angabe der entsprechenden Nummer anwählen. Jeden Programmteil können Sie mit der f2-Taste verlassen. Der Programteil "Thema eingeben" dient dazu ein neues Thema und die dazugehörige erste Stelle einzugeben. Alle weiteren Stellen müssen über den Programmteil "Stelle eingeben" eingetragen werden.
Bearbeitet wird immer das letzte Thema, das in "Thema eingeben" bzw. "Thema suchen" angesprochen wurde. Die Programmteile "Thema löschen", "Stelle eingeben" und "Thema drucken" beziehen sich also immer darauf.
Wollen Sie das aktuelle Thema ändern, so müssen Sie entweder ein neues Thema eingeben oder ein anderes Thema suchen.

Das vorstehende Programm enthält zwar alle notwendigen Programmteile für eine Literaturstellenverwaltung, aber auch nur diese. Ihrer Fantasie bleibt es nun überlassen, das ganze Programm luxuriöser und ausgefeilter zu gestalten. Doch Sie haben sicher erkannt, wie einfach es mit Hilfe der Unterprogramme aus Kapitel 2 geworden ist, sich eine eigene Dateiverwaltung zu schreiben.

Noch ein paar Tips zur Erweiterung :

Wenn Sie im Hauptmenue einen weiteren Programmpunkt eintragen wollen, so tragen Sie den zugehörigen String

einfach in die Tabelle ab 10000 ein und erhöhen die Grenzen in den Zeilen 1040 und 1070. Außerdem müssen Sie noch den Sprung in Zeile 1290 eintragen.

Das Programm läßt sich noch an vielen Stellen verfeinern. Von den Farben, über die Möglichkeit, Eingaben wieder rückgängig zu machen, Suchen mit Jokern und so weiter sind Ihrer Fantasie keine Grenzen gesetzt. Wenn dieses kleine aber feine Programm Ihren Programmierehrgeiz angestachelt hat, so sind wir bereits zufrieden. Wer weiß, was aus diesem Programm vielleicht einmal unter Ihren Händen wird?

4.6 Eine komfortable Adressenverwaltung

Teil 1:

Mit diesem Programm können Sie Ihre Adressen bequem verwalten. Adressen erfassen, nach Adressen suchen und diese für Etiketten passend ausdrucken ist problemlos möglich.
Wenn Sie das Programm starten, erscheint als erstes folgendes Menü auf dem Bildschirm:

```
                 ADRESSVERWALTUNG
      ****************************************
      *                                      *
      *               MENUE                  *
      *                                      *
      *       -1- ADRESSEN EINGEBEN          *
      *                                      *
      *       -2- ADRESSEN AUF DISK          *
      *                                      *
      *       -3- ADRESSEN VON DISK          *
      *                                      *
      *       -4- DRUCKERAUSGABE             *
      *                                      *
      *       -5- BILSCHIRMAUSGABE           *
      *                                      *
      *       -6- ENDE                       *
      *                                      *
      ****************************************
      *                                      *
      *    IHRE WAHL ?                       *
      *                                      *
      ****************************************
```

Sie sehen nun, welche Funktionen das Programm Ihnen zur Verfügung stellt.

Die einzelnen Menüpunkte

-1- ADRESSEN EINGEBEN

In diesem Programmteil geben Sie die Adressen ein. Für die Eingabe ist folgende Form vordefiniert:

ANREDE : (HERR/FRAU/FIRMA o.ä)
NAME :
ADRESSE: (STRASSE oder POSTFACH)
PLZ/ORT:

Nachdem eine Adresse eigegeben wurde, erscheint die Abfrage ob Sie speichern, weiter eingeben oder zurück in das Menü wollen. Dies beantworten Sie mit der entsprechenden Funktionstaste.

-2- ADRESSEN AUF DISK

Hier können Sie aus dem Menü die Adressen auf Diskette abspeichern.

-3- ADRESSEN VON DISK

Mit diesem Programmteil laden Sie auf Diskette gespeicherte Adressen in den C-64 ein.

-4- DRUCKERAUSGABE

Hier haben Sie die Möglichkeit, Adressen auszudrucken. Hierfür stehen Ihnen Zwei Optionen zur Verfügung:
SELEKTION: Beantworten Sie diese Frage mit "j", können Sie nach einem der 4 Adressfelder (Anrede/Name/Adresse/PLZ-Ort) bestimmen, welche Adresse gedruckt werden soll.
Beantworten Sie die Frage mit "n", werden alle gespeicherten Adressen hintereinander ausgedruckt. Das Format entspricht einer Etikettengröße von 33 Spalten und 9 Zeilen

-5- BILDSCHIRMAUSGABE

Diese Programmteil arbeitet grundsätzlich wie unter -4- beschrieben, jedoch erfolgt die Ausgabe auf dem Bildschirm.

Zum Weiterblättern müssen Sie nach jeder Adresse die f1-Taste drücken. Wird das Ende der Datei erreicht, Erscheint die Meldung "ENDE DER DATEN", mit f3 gelangen Sie in das Menü.

-6- ENDE
Mit diesem Programmteil verlassen Sie das Programm.

Teil 2:

Und nun wollen wir uns das Programm teilweise etwas genauer ansehen.
Den zentralen Punkt im Programm stellt das Menü dar. Von hier aus werden die einzelnen Programmfunktionen angesteuert, hierhin kehrt man aus diesen wieder zurück. Wie steuert man nun diese Funktionen, und wie verhindert man Fehleingabe? Eine Antwort auf diese Fragen liefert der erste Teil des Programmes, in den Zeilen 23 - 130.
Als erstes wird ein Menü aufgebaut, indem die einzelnen Funktionen mit Zahlen gekennzeichnet sind, dies erleichtert später die Verzweigung. Bei der Abfrage nach der gewünschten Funktion wird der GET-Befehl für Stringeingabe benutzt. Dies aus zwei Gründen:
a. Das Programm fährt "auf Knopfdruck" weiter, d.h. es wird nur eine Eingabe verlangt.
b. Da der String durch VAL in eine Zahl umgewandelt wird, wird die Fehlerabfrage erleichtert.
Liegt der eingegebene Wert in den entsprechenden Grenzen, kann bequem mit dem ON..GOSUB Befehl in die Unterroutinen verzweigt werden.
Die Zeile 130 mit dem Sprung nach Zeile 5 ist nötig, damit bei der Rückkehr aus den Routinen die Maske neu aufgebaut wird.
Noch ein kleiner Hinweis für die Anwender, die den Datenumfang an Ihre Bedürfnisse anpassen wollen. Die max. Anzahl der Datensätze wird durch eine Variable bestimmt (s. Variablenliste), festgelegt ist sie momentan auf 200. Das

Format des Etikettenpapiers ist auf 33 Spalten auf 9 Zeilen definiert. Wollen Sie hier Änderungen vornehmen, können Sie die Variable für die Spaltenzahl sowie den Ausdruck in den Zeilen 4110 - 4199 verändern.

Variablenliste - Adressenverwaltung

max	Maximale Datensatzzahl
sp	Spaltenzahl der Etiketten
ad$(*,*)	Zweidimensionales Feld für Adressen
w$	Eingabestring für Menü
w	Verzweigungsvariable
a	Datensatznummer
i,j	Laufvariablen
sl	Selektionsfeld
sl$	Selektionsstring
ls	Länge Selektionsstring

Und nun das Programmlisting

```
0 REM *** ADRESSEN V. 1.1 DEZ 83 ***
1 MAX=200 : SP=33
2 DIMAD$(MAX,4)
5 PRINT CHR$(147)+CHR$(142)+CHR$(8)
10 REM ** INITIALISIERUNG BILDSCHIRM :*
20 POKE 53280,2 : POKE53281,2 : PRINT CHR$(144)
22 REM ** AUFBAU MENUE **
23 PRINT SPC(10);CHR$(18)"ADRESSENVERWALTUNG": PRINT
29 PRINT"*****************:********************";
30 PRINT"*"SPC(38)"*"; : PRINT"*"SPC(16)"MENUE"SPC(17)"*";
: PRINT"*"SPC(38)"*";
31 PRINT"*        -1- ADRESSEN EINGEBEN          *"; : PRINT
"*"SPC(38)"*";
32 PRINT"*        -2- ADRESSEN AUF DISK          *"; : PRINT
```

```
"*"SPC(38)"*";
33 PRINT"*      -3- ADRESSEN VON DISK           *"; : PRINT
"*"SPC(38)"*";
34 PRINT"*      -4- DRUCKERAUSGABE              *"; : PRINT
"*"SPC(38)"*";
35 PRINT"*      -5- BILDSCHIRMAUSGABE           *"; : PRINT
"*"SPC(38)"*";
36 PRINT"*      -6- ENDE                        *"; : PRINT
"*"SPC(38)"*";
37 PRINT"****************************************";
38 PRINT"*"SPC(38)"*"; : PRINT"*   IHRE WAHL ?"SPC(24)"*";
: PRINT"*"SPC(38)"*";
39 PRINT"****************************************";
100 GETW$:IFW$=""THEN100
110 W=VAL(W$):IFW<1ORW>6THEN100
120 ONWGOSUB1000,2000,3000,4000,5000,9999
130 GOTO5
1000 A=A+1
1010 PRINT CHR$(147);CHR$(18)"ADRESSEN EINGEBEN  NR.";A:PRI
NT:PRINT:
1020 INPUT"ANREDE : ";AD$(A,1) : PRINT
1021 IF LEN(AD$(A,1)) > SP THEN 1020
1022 INPUT"NAME   : ";AD$(A,2) : PRINT
1023 IF LEN(AD$(A,2)) >SP THEN 1022
1024 INPUT"ADRESSE: ";AD$(A,3) : PRINT
1025 IF LEN(AD$(A,3)) >SP THEN 1024
1026 INPUT"PLZ/ORT: ";AD$(A,4) : PRINT
1027 IF LEN(AD$(A,4)) >SP THEN 1026
1028 PRINT : PRINT CHR$(18)"F1=WEITER"SPC(3)"F3=SPEICHERN"S
PC(3)"F5=MENUE"
1030 GETA$
1040 IFA$=CHR$(133)THEN1000
1050 IFA$=CHR$(134)THENGOSUB2000:RETURN
1060 IFA$=CHR$(135)THENRETURN
1070 GOTO1030
2000 PRINT CHR$(147);CHR$(18)"ADRESSEN AUF DISK":PRINT:PRIN
T
2010 OPEN1,8,15,"S:ADRESSEN.DAT":CLOSE1
2020 OPEN1,8,2,"ADRESSEN.DAT,S,W":PRINT#1,A
2030 FORI=1TOA:FORJ=1TO4
```

```
2040 PRINT"GESSHRIEBEN WIRD ADRESSE NR."I;CHR$(145)
2050 PRINT#1,AD$(I,J):NEXT:NEXT:CLOSE1
2060 RETURN
3000 PRINT CHR$(147);CHR$(18)"ADRESSEN VON DISK":PRINT:PRIN
T
3020 OPEN1,8,2,"ADRESSEN.DAT,S,R":INPUT#1,A
3030 FORI=1TOA:FORJ=1TO4
3040 PRINT"GELESEN WIRD ADRESSE NR."I;CHR$(145)
3050 INPUT#1,AD$(I,J):NEXT:NEXT:CLOSE1
3060 RETURN
4000 PRINT CHR$(147);CHR$(18)"DRUCKERAUSGABE":PRINT:PRINT
4010 OPEN4,4,7
4020 INPUT"SELEKTIEREN J/N";EG$ : IFEG$="J"THENGOSUB4500 :
RETURN
4030 FORI=1TOA : GOSUB4110 : NEXT : CLOSE4 : RETURN
4100 REM ** SUBROUTINE ETIKETTENDRUCK **
4105 PRINT"GEDRUCKT WIRD ADRESSE NR."I;CHR$(145)
4110 PRINT#4
4120 PRINT#4,AD$(I,1)
4130 PRINT#4
4140 PRINT#4,AD$(I,2)
4150 PRINT#4
4160 PRINT#4,AD$(I,3)
4170 PRINT#4,PRINT#4,AD$(I,4)
4190 PRINT#4
4199 RETURN
4500 PRINT CHR$(147);CHR$(18)"SELEKTION" : PRINT : PRINT
4510 PRINT"SELEKTION NACH:"
4511 PRINT"-1- ANREDE"
4512 PRINT"-2- NAME"
4513 PRINT"-3- ADRESSE"
4514 PRINT"-4- PLZ/ORT" : PRINT
4515 PRINT"IHRE WAHL ? ";
4520 GETW$ : IFW$=""THEN4520
4530 W=VAL(W$) : IFW<1ORW>4THEN4520
4540 PRINTW : SL=W
4550 PRINT : PRINT"INHALT DER SELEKTION:" : INPUTSL$ : LS=L
EN(SL$)
```

```
4600 FORI=1TOA
4610 IF LEFT$(AD$(I,SL),LS)=SL$ THENGOSUB4100
4620 NEXT : CLOSE4 :RETURN
5000 PRINT CHR$(147);CHR$(18)"BILDSCHIRMAUSGABE":PRINT:PRIN
T
5020 INPUT"SELEKTIEREN J/N";EG$ : IFEG$="J"THENGOSUB5500 :
RETURN
5030 FOR I=1TOA : GOSUB 5100 : NEXT
5040 PRINT : PRINT"ENDE DER DATEN !!" : PRINT CHR$(18)"=> F
3"
5050 GET F3$ : IFF3$<>CHR$(134)THEN5050
5060 RETURN
5100 PRINT CHR$(147);CHR$(18)"ADRESSE NR.";I : PRINT :PRINT
5110 FOR J=1TO4
5120 PRINT AD$(I,J) : NEXT
5130 PRINT : PRINT CHR$(18)"WEITER MIT F1"
5140 GET F1$ : IF F1$<>CHR$(133)THEN5140
5150 RETURN
5500 PRINT CHR$(147);CHR$(18)"SELEKTION" : PRINT : PRINT
5510 PRINT"SELEKTION NACH:"
5511 PRINT"-1- ANREDE"
5512 PRINT"-2- NAME"
5513 PRINT"-3- ADRESSE"
5514 PRINT"-4- PLZ/ORT" : PRINT
5515 PRINT"IHRE WAHL ? ";
5520 GETW$ : IFW$=""THEN5520
5530 W=VAL(W$) : IFW<1ORW>4THEN5520
5540 PRINTW : SL=W
5550 PRINT : PRINT"INHALT DER SELEKTION:" : INPUTSL$ : LS=L
EN(SL$)
5600 FORI=1TOA
5610 IF LEFT$(AD$(I,SL),LS)=SL$ THENGOSUB5100
5620 NEXT : PRINT : PRINT"ENDE DER DATEN !!" : PRINT CHR$(1
8)"=> F3"
5630 GET F3$ : IFF3$<>CHR$(134)THEN5630
5640 RETURN
9999 PRINT CHR$(147):PRINTTAB(18)"ENDE":CLR:END

READY.
```

5. KAPITEL: DIE VERWENDUNG VON PROGRAMMIERHILFEN AM BEISPIEL VON MASTER 64

Im Verlaufe dieses Buches haben Sie gesehen, daß man sich die Programmierung neuer Anwendungen erheblich erleichtern kann, wenn man über eine entsprechende Programmbibliothek mit Bausteinen und Routinen zu den ständig wiederkehrenden Problemen eines guten Anwendungsprogramms besitzt. Viele dieser Routinen werden Sie sicherlich mit der Zeit selbst entwicklen oder aber aufgrund von Empfehlungen oder sogar entsprechenden Listings in der Fachliteratur und den immer zahlreicher werdenden Computerfachzeitschriften in Ihre Programmbibliothek übernehmen können.

Während für den engagierten Hobbyisten die Entwicklung neuer Routinen und Programmiertricks oft reiner Selbstzweck ist, der ihm auch entsprechenden Spaß macht, richtet der Programmierprofi sein Augenmerk stärker auf die eigentliche Anwendung. Die vielen Routinen, die er für eine effiziente Programmerstellung braucht, sind für ihn reine Werkzeuge, und was liegt näher, als sich derartige Werkzeuge einfach zu kaufen. Das macht zwar nicht so viel Spaß wie die Eigenentwicklung, spart dafür aber eine Menge Zeit und damit letztendlich auch Geld.

Für den Commodore 64 werden mittlerweile eine ganze Reihe verschiedener BASIC-Erweiterungen und Hilfsmittel angeboten, die die Programmerstellung bedeutend erleichtern und komfortabler machen sollen. Stellvertretend für die zahlreichen Hilfsmittel möchten wir Ihnen anhand von MASTER 64 zeigen, welche Möglichkeiten ein solches zugekauftes Programmierwerkzeug bietet.

MASTER 64 ist sehr ausgereift und umfassend, so daß man es ohne weiteres als Programmentwicklungssystem bezeichnen kann. MASTER wurde ursprünglich nach dem Vorbild entsprechender Programmierwerkzeuge im Großrechnerbereich für die großen Commodore Computer der Serie 8000 entwickelt. Es existiert heute in Versionen für die Commodore Computer

8032, 8096, 64, 600 und 700. Alle Versionen sind weitestgehend identisch, so daß Programme, die unter MASTER auf einem Commodore Rechner erstellt wurden, sich leicht auf andere Commodore Rechner übertragen lassen.

MASTER 64 unterstützt viele der Konzepte, die in diesem Buch vorgestellt werden und enthält alle Elemente, die für eine einfache Erstellung anspruchvoller Programme nötig sind, nämlich:

* Bildschirmverwaltung zur raschen Erstellung komfortabler Bildschirmmasken
* ISAM-Dateiverwaltung für schnellen Datenzugriff und effiziente Dateiverwaltung
* Druck-Generator zum einfachen Erstellen und Austesten beliebiger Ausgabemasken
* Programmschutz durch 'nolist'-Modus und individuelle Schlüssel
* Mehrfache genaue Arithmetik, alle Grundrechenarten mit 22 Stellen Genauigkeit
* BASIC-Erweiterungen: Toolkitfunktionen und das komplette BASIC 4.0

Damit Sie einen Eindruck von diesem Programmierwerkzeug bekommen, wollen wir Ihnen die einzelnen Befehle von MASTER 64 einmal vorstellen.

MASTER Bildschirm-Maskengenerator

Wenn Sie die Unterprogramme zur Dateneingabe in diesen Buch gesehen, so haben Sie sicher einen Eindruck davon bekommen, welcher Aufwand für eine komfortable, sicherere, universelle und dazu noch schnelle Eingaberoutine erforderlich ist. MASTER greift nun das Konzept der Bildschirmzonen auf und stellt Ihnen fertige BASIC-Befehle zur Ein- und Ausgabe von Daten zur Verfügung.

Eine Bildschirmzone ist dabei einfach ein Feld auf dem

Bildschirm, das durch seine Position (Zeile und Spalte) und seine Länge gekennzeichnet ist. Mit MASTER können Sie bis zu 127 dieser Zonen definieren, auf die sich dann die Befehle zur Ein- und Ausgabe beziehen.

Bei der Anwendung der neuen Befehle gehen Sie in folgender Reihenfolge vor:

1. Definition der Zone mit Angabe der Position auf dem Bildschirm (Zeile/Spalte), Länge der Zone sowie eine evtl. Formatvorschrift zur automatischen Formatierung von numerischen Daten.
2. Datenanforderung aus einer Zone. In der Zone erscheint der Cursor und Sie können Daten eingeben. Dabei können Sie die Cursortasten beliebig benutzen; ein Verlassen der Zone oder eine Beeinflussung des übrigen Bildschirms ist dabei ausgeschlossen. Die Daten werden mit 'Return' oder anderen selbst zu definierenden Kontrolltasten übernommen.
3. Übernehmen des Zoneninhalts in eine Variable. Mit einem MASTER-Befehl wird der Inhalt einer Zone (bestimmt durch die Zonennummer) einer BASIC-Variablen zugewiesen.
4. Ausgabe in eine Zone. Hiermit können Sie eine Variable oder einen beliebigen Ausdruck formatiert in eine Zone ausgeben.

Die Schritte 2 bis 4 können beliebig oft in einem Programm benutzt werden; die Definition braucht nur einmal zu geschehen.

Zur weiteren Erhöhung des Komforts stehen Ihnen Befehle zum Ziehen von Linien, zur Ausgabe von Daten an beliebige Bildschirmpositionen, sowie zum Löschen, Invertieren und Scrollen (auf und ab) von beliebigen Teilen des Bildschirms zur Verfügung.

Wollen Sie mit mehreren Bildschirmmasken in einem Programm arbeiten, so können Sie mit MASTER zusätzlich virtuelle Bildschirme im Arbeitsspeicher definieren, die Sie schon mit Daten versorgen können, während ein anderer Bildschirm noch

angezeigt wird. Den virtuellen Bildschirm können Sie später mit dem realen Bildschirm austauschen.

Besonders nützlich ist auch die Möglichkeit, komplette Bildschirmseiten einschließlich aller Zonendefinitionen auf Diskette abzuspeichern und von dort wieder zu laden. Sie sparen dadurch in Ihren Programmen viel Speicherplatz, da der Aufbau der Bildschirmmaske durch ein separates Programm geschehen kann, das nur einmal zu laufen braucht. Auch können Sie so die Bildschirmmaske ändern, ohne daß irgendeine Änderung an Ihrem eigentlichen Programm erforderlich ist.

Hier sind nun sämtliche Befehle der MASTER-Bildschirmsteuerung zusammengefasst:

decz n,z,s,l (,ty)(,f$)	Deklariert die Zone Nummer n ab Zeile z, Spalte s mit der Länge l. Optional kann noch ein Typ ty angegeben werden, der bestimmt, ob diese Zone z.B. nur numerische Daten annimmt. Diese Daten können mit dem Formatstring f$ formatiert werden.
reqz n	Mit diesem Befehl wird eine Dateneingabe in Zone n veranlasst.
inz n,a$	Übernahme der eingegebenen Daten in eine Variable
outz n,a$	Ausgabe von Daten in eine Zone
clearz n	Löscht den Inhalt einer Zone
revz n	Invertiert eine Zone auf dem Bildschirm
out a$,z,s	Gibt einen String ab Zeile z, Spalte s aus
clear l,z,s	Löscht ein Feld auf dem Bildschirm
rev z,s,lz,ls	Invertiert ein Bildschirmfeld der Länge lz und der Tiefe ls
scroll z,s,lz,ls,ty	Scrollt ein Bildschirmfeld auf oder ab
tline l,z,s	Zieht eine waagerechte Linie der

	Länge l ab Zeile z, Spalte s
tcol l,z,s	Zieht eine senkrechte Linie der Länge l ab Zeile z, Spalte s
ssave u,fn$	Speichert eine Bildschirmmaske unter dem Namen fn$ auf Diskette ab
sload u,fn$	Lädt eine Bildschirmmaske von Diskette
sset n	Selektiert einen virtuellen Bildschirm
sclear	Löscht den selektierten Bildschirm
scopy	Kopiert einen virtuellen Bildschirm auf den realen Bildschirm
sexch	Tauscht zwei Bildschirminhalte aus

Ein Programmausschnitt zur Eingabe eines Datensatzes könnte z.B. so aussehen:

```
100 decz 1,5,20,6,n: rem zone 1, numerisch
110 decz 2,7,15,25 : rem zone 2, alphanumerisch
120 out "Kundennummer", 5,1 :rem zeile 5, spalte 1
130 out "Name", 7,1
140 reqz 1     : rem eingabe zone 1
150 inz 1,kn : rem kundennummer einlesen
160 reqz 2     : rem eingabe zone 2
170 inz 2,nm$: rem namen einlesen
180 gosub 1000 : rem daten verarbeiten
190 clearz 1,2 : rem zone 1 + 2 löschen
200 goto 140 : rem nächste eingabe
```

MASTER-Druckmasken-Generator

Das gleiche Konzept wie beim Bildschirm-Maskengenerator bietet Ihnen MASTER auch für die Druckausgabe.
Definieren Sie zuerst Ihre Druckseite mit oberen, unterem und linken Rand. Diese Definition wird auf Diskette geschrieben. Nun können Sie analog zum Bildschirm Druckzonen definieren, die wieder durch Zeile, Spalte und Länge sowie Format

bestimmt werden. Zusätzlich können Sie noch konstante Texte einfügen, z.B. für Formulare oder Überschriften. Dies kann auch außerhalb des eigentlichen Anwenderprogramms geschehen. Dort wird dann die Druckmaske nur noch mit den vom Programm anfallenden Daten ausgefüllt, wobei die Ausgabe nicht nur zeilenweise, sondern in beliebiger Reihenfolge geschehen kann, z.B. die letzte Druckzeile zu erst und dann die restlichen Zeilen. Wenn die Maske ausgefüllt ist, kann Sie ein- oder mehrmals auf dem Drucker (zum Austesten auch auf den Bildschirm) ausgegeben werden. Wenn Sie jetzt die Druckfelder wieder löschen, können sie vom Programm wieder neu ausgefüllt werden. Insgesamt stehen Ihnen die gleichen Funktionen wie bei den Bildschirmmasken zur Verfügung (mit Ausnahme der Dateneingabe).

Auch hier stellen wir Ihnen wieder die MASTER-Befehle vor:

pcreate u,fn$,z,s,ro,ru,rl	Deklariert eine Druckseite mit z Zeilen und s Spalten, einem oberen Rand von ro, einem unteren Rand von ru und einem linken Rand von rl.
popen	öffnen einer Druckseite zur Bearbeitung
pdecz n,z,s,l(,ty)(,f$)	Definiert eine Druckzone (analog einer Bildschirmzone)
poutz n,a$	Ausgabe in eine Druckzone
pclearz n	Löscht den Inhalt einer Druckzone
pprint lf	Ausgabe einer Druckseite auf den Drucker
pclear l,z,s	Löschen eines Feldes einer Druckseite
pout a$,z,s	Gibt einen String ab Zeile z, Spalte s aus
pclose	Schließt eine Druckseite
perase	Löscht eine Druckseite

MASTER-ISAM Dateiverwaltung

In der Dateiverwaltung bietet MASTER Ihnen die komfortabelste Möglichkeit, die ISAM-Datei (Index Sequentiell Access Method). Bei der indexsequentiellen Zugriffsmethode wird jeder Datensatz über einen sogenanten Zugriffsschlüssel, den Index, angesprochen. In einer Adressdatei kann dies z.B. der Name sein. Sie brauchen sich also weder die Position des Datensatzes innerhalb der Datei noch eine Datensatznummer zu merken - der Aufruf geschieht einfach über den Namen. Das Anlegen einer ISAM-Datei geschieht menuegesteuert über ein Datei-Reservierungs-Programm, das alle Parameter der Datei abfragt: Datensatzlänge bis max. 254 Zeichen, Schlüssellänge bis zu 30 Zeichen sowie max. Anzahl von Datensätzen. Hinsichtlich der Dateigröße besteht keine Einschränkung, eine MASTER-Datei kann über die ganze Diskette gehen.

Folgende Befehle für ISAM-Dateien stehen Ihnen zur Vefügung:

iopen lf,u,fn$	öffnet eine MASTER-ISAM-Datei mit der logischen Dateinummer lf auf dem Gerät u (normalerweise 8) mit dem Namen fn$
iwrite lf,rc$	Schreibt einen neuen Datensatz rc$ in die Datei
iread lf,ky$,rc$	Liest den zum Zugriffsschlüssel ky$ gehörenden Datensatz in die Variable rc$
iexist lf,ky$	Testet, ob zum Zugriffschlüssel ky$ ein Datensatz existiert
iupdate lf,rc$	Schreibt einen geänderten Datensatz rc$ in die Datei zurück
idelete lf,ky$	Löscht einen Datensatz
istart lf,ky$,mk$,rc$	Startet das Lesen in aufsteigender Reihenfolge. Dabei ist mk$ eine Auswahlmaske, die nur bestimmte Datensätze selektiert.
istart-lf,ky$,mk$,rc$	Wie oben, jedoch in absteigender Reihenfolge
inext lf,mk$,rc$	Liest nächsten Datensatz in Schlüssel-

	reihenfolge
inext-lf.mk$.rc$	Wie oben, jedoch in absteigender Reihenfolge
iadd lf.rc$	Schreibt einen Datensatz ohne automatisches Einfügen des Zugriffsschlüssels
ivalidate lf	Einfügen des Index der mit iadd geschriebenen Datensätze
irestore lf	Positionierung auf den zuerst geschriebenen Datensatz
idata lf,mk$,rc$	Lesen der Datensätze in Eingabereihenfolge
iupgrade lf	Sichern des Index auf Diskette
iclose lf	Schließt eine MASTER-ISAM-Datei

Bei jeder Dateioperation wird eine Statusvaribale 'ok' gesetzt, die Sie über das ordnungsgemäße Ausführen des Befehls informiert. MASTER kann Ihre Datensätze packen, d.h. alle überflüssige (redundante) Information wird entfernt, so daß sich eine Einsparung von ca. 10 - 50% des Datensatzes ergibt; dadurch werden effektive Datensatzlängen von bis zu ca. 400 Zeichen möglich.
Auch die Datensicherheit wird nicht vernachlässigt: Sollte der Index auf der Diskette zerstört sein, steht ein Programm zur Verfügung, daß die komplette Datei wieder rekonstruiert.

Auch hier wieder ein typischen Programmbeispiel mit einer MASTER-ISAM-Datei, das nach einem Kundennamen fragt und aus der ISAM-Datei den entsprechenden Satz liest.

```
100 iopen 1,8, "0:kunden"
110 reqz 1 : inz1, kn$ : rem kundenname
120 iread 1, kn$, da$  : rem datensatz lesen
130 out da$, 10,1 : rem datensatz am bildschirm anzeigen
140 clearz 1 :rem eingabefeld löschen
150 goto 110
```

Außerdem hat MASTER noch folgende BASIC-Erweiterungen: Mehrfach genaue Artihmetik, alle Grundrechenarten können mit 22 Stellen ausgeführt werden. Ebenso existieren Befehle, die den Direktzugriff auf Diskette vereinfachen:

uplow	Vertauschen von Groß- und Kleinbuchstaben
datepack	Packen des Datums und Prüfung auf Gültigkeit
dateunpack	Entpacken des Datums
goto zn	Berechnetes goto, erlaubt berechnete Sprünge und kann z.B. umfangreiche ON..GOTO Sprungleisten ersetzen
gosub zn	Berechnetes gosub
madd n1$,n2$,n3$	Addition mit 22 Stellen
msub n1$,n2$,n3$	Subtraktion mit 22 Stellen
mmul n1$,n2$,n3$	Multiplikation mit 22 Stellen
mdiv n1$,n2$,n3$	Division mit 22 Stellen
creatst a$,l(,a)	Erzeugt einen String der Länge l mit dem ASCII-Kode a
putst a$:ty,b$,p,l	Packen von Daten
getst a$:ty,b$,p,l	Entpacken von Daten
inblock tr,se(,d)(,u)	Liest Track tr Sektor se von Diskette in einen Puffer
putblock tr,se(,d)(,u)	Schreibt einen Block aus einem Puffer auf Diskette
takbuf a$,p,l	Transfer Puffer-String
putbuf a$,p,l	Transfer String-Puffer
hunt chr$(a),a$,p	Suchen von a in a$
nolist a$,d(d)	Abspeichern im nolist-Modus, schützt ein Programm vor unberechtigtem Auflisten.
onerror zn	Abhandlung von Diskettenfehlern

Hier noch ein paar Bemerkungen zum Packen von Daten: MASTER kann alphanumerische Strings sowie Fließ- und Festkommazahlen

packen. Dies geschieht automatisch mit dem Befehl putst zum Packen und getst zum Entpacken. Dabei wird je nach Typ eine Platzersparnis von bis zu ca. 50 % erreicht.

MASTER 1 enthält weiterhin einen sehr nützlichen Programmer's Toolkit, wie wir ihn in diesen Buch häufig erwähnt haben, mit den Befehlen:
auto, delete, renumber, dump, if..then..else, hardcopy, trace on..off
Diese Befehle erleichtern Ihnen Eingabe, Editieren und Austesten Ihrer Programme.

Des weiteren stehen Hilfsfunktionen für ISAM-Dateien zur Verfügung, z. B. übertragen von sequentiellen Dateien in ISAM-Dateien und umgekehrt sowie Dump-Funktionen für ISAM-Dateien.

Für die kommerzielle Nutzung Ihrer Programme ist der Listschutz durch den 'nolist'-Befehl sehr nützlich.
Mit MASTER geschriebene Programme sind auf allen Commodore-Rechnern mit MASTER einsetzbar. Um die volle Kompatibilität der 64er Programme auf den anderen Rechnern zu garantieren, enthält MASTER 64 noch das komplette BASIC 4.0 mit den komfortablen Diskettenbefehle, wie z.B. dload, dsave, scratch, record usw. Damit können Sie auch ohne Klimmzüge relative Dateien auf der VC-1541 benutzen.

5.2 STRUKTO 64

In Kapitel 1 dieses Buches wurde das strukturierte Programmieren erläutert. Wir haben gesehen, daß es grundsätzlich möglich ist, mit BASIC strukturierte Programme zu schreiben, jedoch ist BASIC nicht geeignet, diese Programmstrukturen übersichtlich darzustellen und läßt außerdem immer noch ein "Verletzen" der Programmstruktur zu, was nicht gerade zu einer logisch korrekten Programmierung beiträgt. Warum sollte es also nicht möglich sein, die übersichtliche Programmstruktur von Sprachen wie Pascal mit der Einfachheit der System-Bedienung von BASIC zu vereinen?

Dies wurde durch STRUKTO 64 erreicht:
STRUKTO 64 ist eine strukturierte Programmiersprache, mit der die in Kapitel 1.6 dargestellten Nassi-Shneidermann Diagramme direkt in den Programmtext umgesetzt werden können. Die so erstellten Programme sind danach genauso übersichtlich wie die Struktogramme selbst, was durch ein automatisches Einrücken erreicht wird. Die Bedienung des Systems ist jedoch weiterhin genauso einfach wie die von BASIC und wird sogar noch durch einen integrierten Toolkit erleichtert. Die Programmzeilen sind weiterhin mit Nummern versehen, jedoch dienen diese nicht mehr als Sprungadressen für GOTO- und GOSUB-Befehle, sondern nur noch als Identifikation der jeweiligen Zeile beim LISTen bzw. Eingeben von Zeilen.

Nun aber genug der Theorie. Hier sind die von STRUKTO 64 verwendeten Kontrollstrukturen: (vgl. Kapitel 1.6.)

1. Der Verarbeitungsschritt

Dieser stellt eine einfache Anweisung dar, die genau dann ausgeführt wird, wenn der Interpreter an der entsprechenden Programmzeile "ankommt".

Er sieht in STRUKTO 64 genauso aus wie in BASIC:

```
140 print a$," Datum:";dt$
```

2. Der Unterprogrammaufruf

Es gibt in STRUKTO 64 zwei Arten von Unterprogrammen, nämlich solche mit Nummern oder solche mit Namen. Ein Unterprogramm mit Nummer sieht dann folgendermaßen aus:

```
3000 begin 6
3010   input "6. Eingabe:";a$(6)
3020   print
3030 end
```

Und so ein Unterprogramm mit Namen:

```
4000 entry linie zeichnen
4010   x=cos(t)*80+90
4020   y=sin(t)*80+80
4030   line 0,0;x,y
4040 exit
```

Aufgerufen werden diese Unterprogramme durch den Befehl gosub 6 (bei dem Unterprogramm mit Nummer) bzw. pass linie zeichnen (bei dem Unterprogramm mit Namen).

3. Die Zweifachverzweigung

Die Zweifachverzweigung wird in STRUKTO 64 durch die if-Anweisung verwirklicht. Beispiel:

```
100   if a=0
```

```
110     print "a gleich 0"
120    ifnot
130     print "a ungleich 0"
140    ifend
```

Die Befehle zwischen if und ifnot werden genau dann ausgeführt, wenn die Bedingung erfüllt ist, sonst die zwischen ifnot und ifend. Das ifnot kann auch fehlen, dann wird der Programmteil übersprungen, wenn die Bedingung nicht erfüllt ist. In jedem Fall laufen die beiden Programmzweige hinter dem ifend wieder zusammen.

4. Die Mehrfachverzweigung

Diese wird in STRUKTO 64 durch den case-befehl verwirklicht. Nach dem "case" folgt ein beliebiger Ausdruck, danach eine Aufzählung von Programmzweigen, die dann durchlaufen werden, wenn einer der am Beginn des Zweiges angegebenen Ausdrücke mit dem bei case übereinstimmt. Alle Programmzweige laufen dann beim "caseend" wieder zusammen
Beispiel:

```
150    input "Zahl zwischen 0 und 10 eingeben:";a
160    case a
170    of 0,2,4,6,8,10
180     print "a ist gerade"
190    of 1,3,5,7,9
200     print "a ist ungerade"
210    ofrest
220     print "falsche Eingabe"
230    caseend
```

5. Die Schleife

In STRUKTO 64 gibt es drei verschiedene Schleifentypen:

a) Laufbedingung am Anfang der Schleife:

Diese Schleife kann auch nullmal durchlaufen werden und sieht so aus:

```
100 while Laufbedingung
110   :
120   :
130   :
140 whileend
```

Die Befehle zwischen while und whileend werden solange ausgeführt, wie die Laufbedingung erfüllt ist.

b) Abbruchbedingung am Schluß der Schleife

Diese Schleife wird mindestens einmal durchlaufen und sieht zum Beispiel so aus:

```
100 repeat
110   input "0 oder 1 eingeben:";x
120 until x=0 or x=1
```

c) Abbruchbedingungen an beliebigen Stellen innerhalb der Schleife

Bei dieser Schleife wird der Anfang durch "loop" und das Ende durch "loopend" gekennzeichnet. In der Schleife können beliebig viele Abbruchbedingungen untergebracht werden, die mit "on Bedingung leave" gekennzeichnet sind. Eine solche Schleife sieht dann so aus:

```
400   loop
410     ...
```

```
420       ...
430    on Bedingung1 leave
440       ...
450       ...
460    on Bedingung2 leave
470    loopend
```

d) Schleife mit Zählvariable

Eine solche Schleife wird in STRUKTO 64 durch for und forend gekennzeichnet und entspricht in Aufbau und Funktion der FOR/NEXT-Schleife in BASIC.

Dies sind also die Kontrollstrukturen von STRUKTO 64. Aber STRUKTO 64 bietet gleichzeitig eine umfangreiche Befehlserweiterung und vereinfacht die Erarbeitung umfangreicher Programme. Es sollen nun einige Besonderheiten von STRUKTO 64 dargestellt werden.

DAS TOOLKIT

Folgende Befehle erleichtern das Eingeben, Verbessern, Testen, Laden und Speichern von Programmen:

auto — gibt beim Eingeben von Programmen die Zeilennummern vor.
ren — nummeriert die Zeilen neu durch.
indent — rückt die Kontrollstrukturen ein.
trace — zeigt die Nummer der aktuellen Zeile an.
& — hebt den trace- bzw. auto-Modus auf.
find — sucht nach besimmten Stellen im Programmtext.
cont — setzt das durch (stop) abgebrochene Programm fort.
new — löscht das gesamte Programm.
del — löscht bestimmte Programmteile.
old — holt, falls möglich, ein gelöschtes Programm zurück.

edit   gibt eine Programmzeile zum Verbessern aus.

Außerdem können die Funktionstasten jeweils vierfach mit einer beliebigen Tastenfolge (bis zu 10 Tasten) belegt werden, so daß das Drücken einer Funktionstaste das Drücken von mehreren anderen Tasten ersetzt.

Dies sind die zusätzlichen Befehle, die den Umgang mit der Floppy oder dem Drucker erleichtern:

append hängt ein Programm an das geladene an.
dir   zeigt das Inhaltsverzeichnis der Diskette an.
!   liest den Fehlerkanal der Floppy.
disk   gibt einen Befehl an die Floppy.
mon   gibt die Ein- und Ausgaben von außen auf den Bildschirm

Hierbei braucht an Befehle wie load, save, verify oder append nicht jedes mal die Geräte-Nummer angehängt werden. Es genügt, diese einmal anzugeben, und sie bleibt solange gespeichert, bis eine neue angegeben wird. Die augenblicklich gespeicherte Geräte-Nummer ist jederzeit in der Variablen dn% verfügbar.

Der disk-Befehl gibt einen Befehl an die Floppy weiter und ersetzt dadurch die BASIC-Befehle
OPEN 15,8,15,"...":CLOSE 15 durch disk "...".
Dabei kann der Rechner bereits weiterarbeiten, während die Floppy den Befehl ausführt.

Eine weitere Besonderheit von STRUKTO 64 ist das problemlose Aneinanderfügen von Programmen. Nehmen wir einmal folgenden, nicht selten auftretenden Fall an: Sie haben mehrere gute Programme geschrieben, die Sie nun gerne zu einem Programm zusammensetzten wollen, in dem die ursprünglichen Programme als Unterprogramme aufgerufen werden. Dabei können Sie mit STRUKTO 64 folgendermaßen vorgehen:
Sie kennzeichnen den Anfang der jeweiligen Programme mit

entry name und das Ende mit exit. Dann löschen Sie den Programmspeicher und schreiben das Hauptprogramm, daß die anderen als Unterprogramme aufruft. Anschließend hängen Sie die Unterprogramme mit append an - fertig. Dies können Sie zwar auch mit einem geeigneten Programm in BASIC machen, aber nehmen wir einmal an, in einigen dieser Unterprogramme werden datas verwendet. Da diese datas in BASIC stets vom Programmanfang zum Ende gelesen werden, kann es dabei zu Mißverständnissen kommen. In STRUKTO 64 wird dieses Problem dadurch gelöst, daß jedes Unterprogramm einen eigenen data-Satz besitzen kann, der nur von diesem Unterprogramm gelesen werden kann, und den data-read-Vorgang im übergeordneten Hauptprogramm nicht beeinflußt.

Wenn nun die einzelnen Programme zu lang sind, um gleichzeitig im Hauptspeicher Platz zu finden, so kann durch einen load-Befehl im Programm ein anderes Programm direkt geladen und sofort gestartet werden. Bei umfangreichen Programmen kann zum Beispiel das Hauptmenue ein kleines Programm sein, daß andere Programme nachlädt und startet, die dann nach ihrem Durchlauf wieder das Menue-Programm laden und starten.

5.3 DAS STRUKTO 64-GRAPHIKKONZEPT

Die Mehrheitsentscheidung

Wenn Sie mit dem Commodore 64 farbige Graphiken entwerfen wollen, so werden Sie auf das Problem stoßen, daß in einem Feld aus 8 mal 8 Punkten in der hochauflösenden Graphik (dieses entspricht einem Zeichen auf dem normalen Bildschirm) immer nur höchstens zwei Farben gleichzeitig dargestellt werden können. Es ist also z. B. nicht möglich, in einem solchen Feld gleichzeitig eine rote und eine blaue Linie , die sich schneiden, auf schwarzem Grund darzustellen. Auf Grund dieser Einschränkung ist es in den meisten für den Commodore 64 gebräuchlichen Graphikerweiterungen üblich, daß zum Zeichnen eines Punktes zwei Farben und zusätzlich eine Information darüber, ob der Punkt "gesetzt" oder "gelöscht" werden soll, angegeben werden muß.

Dies bringt in vielen Fällen, vor allem, wenn die Lage bestimmter Linien zueinander vor dem Programmablauf unbekannt ist, einen großen Programmieraufwand mit sich. Das STRUKTO 64-Konzept läßt sich nun auf folgende Weise kurzfassen:
Sie programmieren so, als könne der Commodore 64 Farben in beliebigen Kombinationen darstellen, und der Interpreter macht dann das bestmögliche daraus. Und das funktioniert so: Nehmen wir an, in einem Feld aus 8 mal 8 Punkten sind bereits zwei Farben dargestellt (eine blaue Linie von links nach rechts auf schwarzem Grund). Nun sollen in diesem Feld noch einige Punkte in einer dritten Farbe gezeichnet werden (eine rote Linie von oben nach unten). Theoretisch müßten in diesem Feld jetzt drei Farben dargestellt werden (schwarz, rot, blau), was aber praktisch nicht möglich ist. Da die soeben gezeichneten Punkte in jedem Fall die gewählte Farbe (rot) haben sollen, beiben zwei mögliche Entscheidungen:
Entweder wird die eine schon vorhandenen Farbe (schwarz)

oder die andere (blau) durch die dritte (rot) ausgetauscht. STRUKTO 64 entscheidet sich nun je nachdem, in welchem Fall weniger Information zerstört wird, d. h. es wird in jedem Fall so wenig Farbe verändert, wie nötig (die blaue Linie wird im Bereich dieses Feldes rot).

In der Praxis ist dadurch das erstellen farbiger Graphiken sehr einfach, da die Besonderheiten des Commodore 64 nicht beachtet werden brauchen, denn die geringfügigen Farbveränderungen machen sich kaum bemerkbar. Wer mehr Wert auf korrekte Farbe legt, kann dann noch die Multicolor-Graphik verwenden, bei der die Auflösung zwar nur halb so groß ist, dafür aber in einem oben beschriebenen Feld 4 Farben dargestellt werden können. Auch hier fällt der Interpreter wieder eine Mehrheitsentscheidung, wenn der Programmierer versucht, mehr als 4 Farben darzustellen.

Das Fenster

In der Graphik lassen sich beliebige Rechtecke aus 8 mal 8-Punkte-Feldern als Fenster definieren. Diese Fenster können dann mit einer einzigen Farbe ausgefüllt oder in beliebige Richtungen "gescrollt" werden, d.h. der Inhalt des Feldes wird nach oben, unten, rechts oder links verschoben. Dabei ist es auch möglich, die am einen Ende des Fensters veschluckten Zeilen bzw. Spalten am anderen Ende wieder zum Vorschein zu bringen. Auch ein Austauschen von Farben in dem Fenster ist möglich.
Dies ist besonders nützlich zur Darstellung bewegter Graphiken oder zur Hervorhebung von Daten durch blinkenden Hintergrund.

Text in der Graphik

Mit STRUKTO 64 ist es möglich, innerhalb des Fensters Text darzustellen, indem die Graphik als Gerät Nr. 17

angesprochen wird (wie Drucker = 4). Beim Versuch, aus dem Fenster hinaus zu drucken, wird automatisch ein entspechender Scroll ausgeführt, so daß man die Graphik letztlich genauso benutzen kann, wie den Text-Bildschirm. Dabei besteht der Vorteil, daß in der Graphik gleichzeitig beide Zeichensätze dargestellt werden können, auf dem Textbildschirm jedoch immer nur einer. Außerdem können in der Graphik doppelt-breite oder doppelt-hohe Zeichen dargestellt werden.
Zeichensätze können in Strukto 64 auch von Diskette geladen oder dorthin abgespeichert werden. Auch das Herstellen eigener Zeichensätze ist möglich.

Sprites

Die Sprites können mit STRUKTO 64 direkt vom Bildschirm eingelesen werden. D. h. Sie "zeichnen" die Formen auf dem Bildschirm, sei es im direkt-Modus oder mit print-Anweisung, und geben dann den shget-Befehl zum Einlesen einer Formtabelle. Sie können gleichzeitig 16 solcher Tabellen speichern, ohne selbst den Speicherplatz dafür zu organisieren. Mit dem Befehl spdef a,b,f,... teilen Sie dem Computer nun mit, daß das Sprite Nummer a die unter Nummer b eingelesene Form und die Farbe f bekommen soll, außerdem kann es in x- oder y-Richtung gestreckt werden. Danach können Sie das Sprite durch den spput-Befehl auf dem Bildschirm bewegen oder mit spon/spoff ein und ausschalten. Auch ein Abspeichern der Formtabellen ist möglich.

MUSIK

STRUKTO 64 bietet die Möglichkeit, bis zu dreistimmige Melodien mit bis zu 500 Noten pro Stimme einzulesen und diese dann, während das Programm weiterläuft oder der

Benutzer im Direkt-Modus arbeitet, abzuspielen. Die Noten werden dabei direkt als solche eingegeben, also zum Beispiel "c64" für ein c der Länge 64 = 1 Sekunde = ganze Note. Die Noten werden dabei getrennt abgespeichert und belegen keinen Programmspeicherplatz.

BEISPIELPROGRAMM

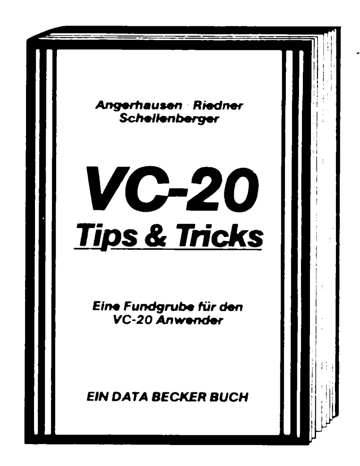
Um den logisch strukturierten Programmaufbau von STRUKTO 64 möglichst deutlich zu machen, haben wir als Beispiel für ein kleines STRUKTO-Programm ein solches aus dem Bereich der numerischen Mathematik gewählt: Berechnung von Primzahlen nach dem Sieb-Verfahren.

Dabei werden eine vorgegebene Anzahl von Zahlen zunächst als Primzahlen gekennzeichnet. Dann wird die kleinste Zahl als Primzahl ausgegeben und alle Vielfachen dieser Zahl werden "aussortiert". Die kleinste noch nicht aussortierte Zahl ist dann wieder eine Primzahl, die wieder ausgegeben wird u.s.w. Um alle Primzahlen bis 1000 zu berechnen, benötigt dieses Programm etwa 28 Sekunden, ist also wesentlich schneller als ein Programm mit dem einfachen Divisions-Test-Verfahren.

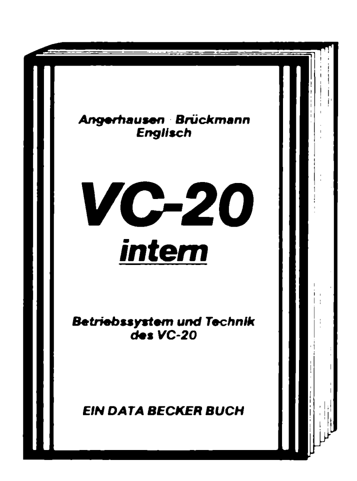
```
100    rem  programm zur primzahl-
110    rem  berechnung nach dem
120    rem  sieb-verfahren:
130    rem  die vielfachen einer
140    rem  gefundenen primzahl
150    rem  werden aussortiert.
160    rem
170    rem
180    repeat
190      clr
200      print"";
210      centre "Primzahlen"
220      print
230      repeat
240        input "von 2 bis ";b
250      until b=int(b) and b<=15000 and b>2
260      dim a%(b/2)
270      print "      2.";
280      for f=3 to b step 2
290        rem
300        rem  zählen von 3 bis b
310        rem  gerade zahlen weglassen
320        rem
330        if a%(f/2)=0
340          rem
350          rem f ist eine primzahl
360          rem formatierte ausgabe
370          rem
380          printf;
390          i=3*f
400          rem
410          rem ungerade vielfache
420          rem aussortieren
430          rem
440          while i<b
450            a%(i/2)=1
```

```
460            i=i+2*f
470          whileend
480        ifend
490      forend
500      print
510      print "nochmal? (j/n)";
520      repeat
530        get a$
540      until a$="j" or a$="n"
550    until a$="n"
560 end
```

DATA BECKER BÜCHER



Die überarbeitete und erweiterte 2. Auflage von VC-20 TIPS & TRICKS enthält eine detaillierte Beschreibung der Programmierung von Sound und Graphik des VC-20, mehr über Speicherbelegung, Speichererweiterung und die optimale Nutzung der einzelnen Speichermodule, BASIC-Erweiterungen zum Eintippen, umfangreiche Sammlung von Poke's und anderen nützlichen Routinen, zahlreiche interessante Beispiel- und Anwendungsprogramme, komplett dokumentiert und fertig zum Eintippen (z. B. Spiele, Funktionenplotter, Graphik Editor, Sound Editor) und vieles andere mehr. VC-20 TIPS & TRICKS ist eine echte Fundgrube für jeden VC-20 Anwender. VC-20 TIPS & TRICKS, 2. Auflage 1983, ca. 230 Seiten, DM 49,–.



Die überarbeitete und erweiterte 2. Auflage von VC-20 INTERN beschäftigt sich detailliert mit Technik und Betriebssystem des VC-20 und enthält ein ausführlich dokumentiertes ROM-Listing, die Belegung der ZEROPAGE und anderer wichtiger Bereiche, übersichtliche Zusammenfassungen der Routinen des BASIC-Interpreters und des VC-20 Betriebssystems, eine Einführung in die Programmierung in Maschinensprache, eine detaillierte Beschreibung der Technik des VC-20 und als Clou drei Original COMMODORE Schaltpläne zum Ausklappen! Damit ist VC-20 INTERN für jeden interessant, der sich näher mit Technik und Maschinenprogrammierung des VC-20 auseinandersetzen möchte. VC-20 INTERN, 2. Auflage 1983, ca. 230 Seiten, DM 49,–.

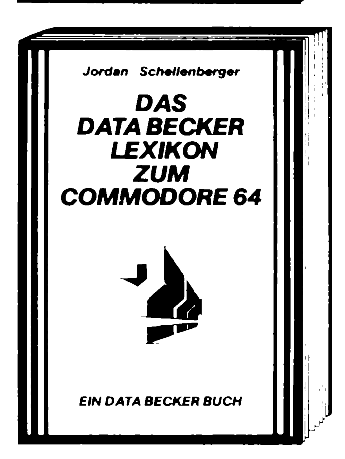
DATA BECKER BÜCHER



Das sollte Ihr erstes Buch zum COMMODORE 64 sein: 64 FÜR EINSTEIGER ist eine sehr leicht verständliche Einführung in Handhabung, Einsatz, Ausbaumöglichkeiten und Programmierung des COMMODORE 64, die keinerlei Vorkenntnisse voraussetzt. Schritt für Schritt führt das Buch Sie in die Programmiersprache BASIC ein, wobei Sie nach und nach eine komplette Adressenverwaltung erstellen, die Sie anschließend nutzen können. Zahlreiche Abbildungen und viele Anregungen zum sinnvollen Einsatz des COMMODORE 64.
Das Buch ist sowohl als Einführung als auch als Orientierung vor dem 64er Kauf gut geeignet. Ca. 200 Seiten, DM 29,–.

Plenge
DAS GRAFIKBUCH ZUM COMMODORE 64
EIN DATA BECKER BUCH

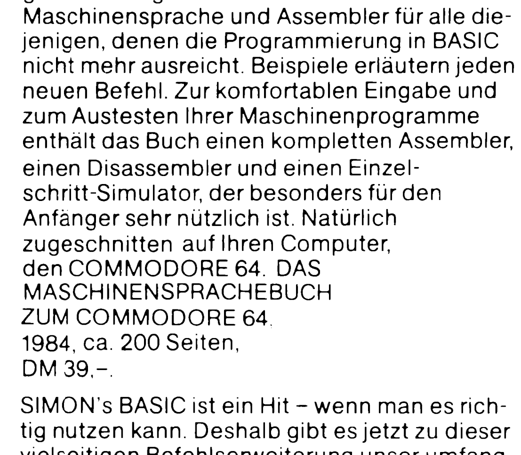
Graphik ist eine der Hauptstärken des COMMODORE 64. Mit diesem neuen Buch lernen Sie, wie Sie die graphischen Fähigkeiten programmtechnisch optimal nutzen. Der Inhalt reicht von den Grundlagen der Graphikprogrammierung über das Erzeugen einfacher Figuren, die Arbeit mit Sprites, Zeichensatzprogrammierung, Hardcopy und IRQ-Handhabung bis hin zur Funktionendarstellung, Laufschrift, Statistik, 3-D, CAD, den Geheimnissen der Actionsspiele und Lightpenanwendungen. Zahlreiche Beispielprogramme ergänzen dieses Buch, das die faszinierende Computertechnik jedermann zugänglich macht.
Ca. 250 Seiten, DM 39,–.



Diese neue, umfangreiche Programmsammlung hat es in sich. Über 50 Spitzenprogramme für den COMMODORE 64 aus den unterschiedlichsten Bereichen, vom Superspiel („Senso", „Pengo") über Graphik- und Soundprogramme (zum Beispiel „Fourier 64" oder „Orgel") sowie Utilities („Sort") bis hin zu Anwendungsprogrammen wie „Videothek" oder „Finanzbuchhaltung". Der Hit sind zu jedem Programm aktuelle Programmiertips und Tricks der einzelnen Autoren zum Selbermachen. Also – nicht nur abtippen, sondern auch dabei lernen und wichtige Anregungen für die eigene Programmierung sammeln.
Ca. 250 Seiten, DM 49,–.

DATA BECKER BÜCHER





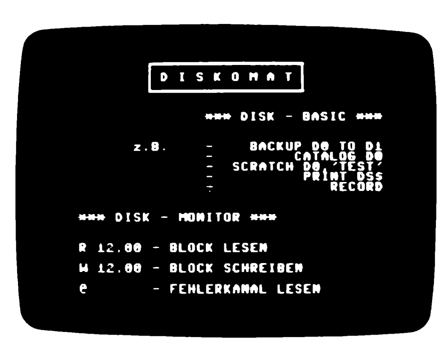
Eine leicht verständliche Einführung in die Programmierung des COMMODORE 64 in Maschinensprache und Assembler für alle diejenigen, denen die Programmierung in BASIC nicht mehr ausreicht. Beispiele erläutern jeden neuen Befehl. Zur komfortablen Eingabe und zum Austesten Ihrer Maschinenprogramme enthält das Buch einen kompletten Assembler, einen Disassembler und einen Einzelschritt-Simulator, der besonders für den Anfänger sehr nützlich ist. Natürlich zugeschnitten auf Ihren Computer, den COMMODORE 64. DAS MASCHINENSPRACHEBUCH ZUM COMMODORE 64. 1984, ca. 200 Seiten, DM 39,–.

SIMON's BASIC ist ein Hit – wenn man es richtig nutzen kann. Deshalb gibt es jetzt zu dieser vielseitigen Befehlserweiterung unser umfangreiches Trainingsbuch, das Ihnen detailliert den Umgang mit den über 100 Befehlen des SIMON's BASIC erklärt. Ausführliche Darstellung aller Befehle (auch der, die nicht im Handbuch stehen!) Natürlich auch mit allen Macken und Hinweisen, wie man diese umgeht. Dazu zahlreiche Beispielprogramme und interessante Programmiertricks. Nach jedem Kapitel Testaufgaben zum optimalen Selbststudium. Dieses Buch sollte jeder SIMON's BASIC Anwender unbedingt haben! Das TRAININGSBUCH ZUM SIMON's BASIC, 1984, ca. 300 Seiten, DM 49,–.



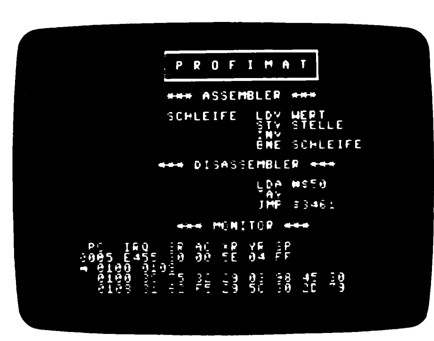
Darauf haben Sie gewartet: Endlich ein Buch, das Ihnen ausführlich und verständlich die Arbeit mit der Floppy VC-1541 erklärt. DAS GROSSE FLOPPY BUCH ist für Anfänger, Fortgeschrittene und Profis gleichermaßen interessant. Sein Inhalt reicht von der Programmspeicherung bis zum DOS-Zugriff, von der sequentiellen Datenspeicherung bis zum Direktzugriff, von der technischen Beschreibung bis zum ausführlich dokumentierten DOS Listing, von den Systembefehlen bis zur detaillierten Beschreibung der Programme der Test/Demodiskette. Exakt beschriebene Beispiel- und Hilfsprogramme ergänzen dieses neue Superbuch. Mit dem GROSSEN FLOPPY-BUCH meistern Sie auch Ihre Floppy. DAS GROSSE FLOPPY BUCH, 1983, ca. 320 Seiten, DM 49,–.

Die neuen
DATA BECKER PROGRAMME



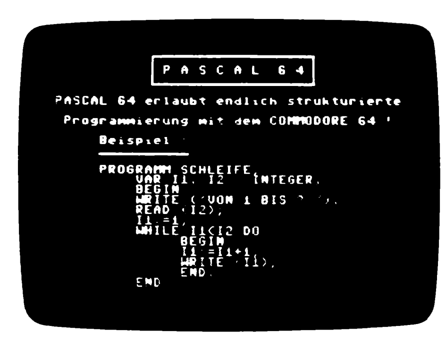
DISKOMAT

Dieses neue Spitzenpaket hilft Ihnen, mehr aus Ihrer Floppy zu machen, mit SUPERTWIN, dem Steuerprogramm, das zwei VC-1541 wie ein Doppellaufwerk verwaltet, mit DISK-BASIC, den Diskettenbefehlen des BASIC 4.0, mit denen Sie eine komplette Diskette oder Auszüge mit einem Befehl kopieren können und mit eine komfortablen DISK-MONITOR, mit dem Sie den Aufba der Diskette erforschen und manipulieren können. Alle zusammen für nur DM 99,-.



PROFIMAT

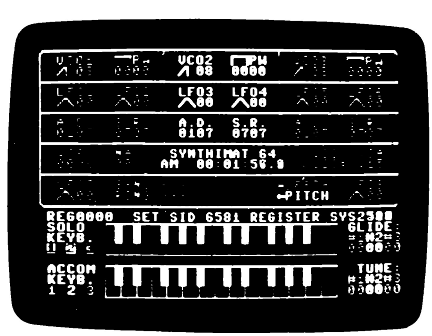
Ein Spitzenpaket für Maschinenspracheprogrammierer PROFIMAT enthält nicht nur unseren komfortablen Maschinensprache Monitor PROFI-MON, sondern auc PROFI-ASS, einen sehr leistungsfähigen Assembler für den COMMODORE 64. PROFI-ASS bietet unter andere formatfreie Eingabe, komplette Assemblerlistings, ladba Symboltabellen (Labels), verschiedene Möglichkeiten z Speicherung des erzeugten Maschinencodes, redefinie bare Symbole, eine Reihe von Pseudo-Codes (Assemble anweisungen), bedingte Assemblierung und die Möglic keit zur Erzeugung von Assemblerschleifen. PROFIMA kostet komplett mit ausführlichem Handbuch nur DM 99



PASCAL 64

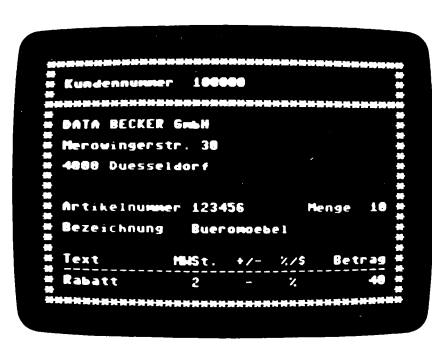
Jetzt können Sie die beliebte Sprache PASCAL auch a dem COMMODORE 64 einsetzen. PASCAL 64 ist ein leistungsfähiger PASCAL-Compiler, der nicht nur den Befehlssatz des Standard PASCAL unterstützt, sonder auch die hochauflösende Graphik und die Sprites des Commodore 64. Ein-/Ausgabe über Diskette und Druck sowie REAL und INTEGER Arithmetik. Unterprogramm aus Ihrer eigenen Programmbibliothek können vor den Compilieren in Ihr Hauptprogramm mit eingebunden werden. PASCAL 64 ist sehr schnell, da echter Maschine code erzeugt wird. PASCAL 64 kostet komplett mit aus führlichem Handbuch nur DM 99,-.

Die neuen
DATA BECKER PROGRAMMI



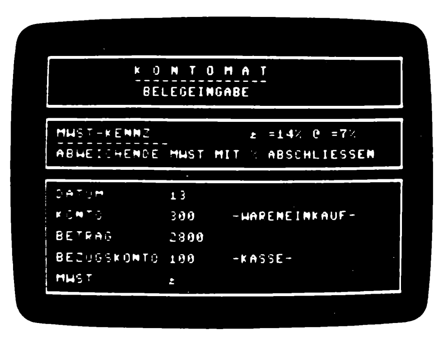
SYNTHIMAT

Mit diesem Superprogramm verwandeln Sie Ihren 64e
einen professionellen, polyphonen, dreistimmigen
Synthesizer, mit dem Sie über die Tastatur ganze Akkc
spielen können. Zu den unglaublich vielen Möglichke
dieses Programms gehört auch die Bandaufnahme/
-wiedergabe direkt auf bzw. von Diskette. SYNTHIMA
stellt gleichzeitig den Synthesizer optisch dar. Sämtli
Module sind farblich gekennzeichnet und übersichtlic
angeordnet. Es ist ein Leichtes, mit SYNTHIMAT sämtli
Klangeigenschaften verschiedener Musikinstrumente
imitieren, aber auch völlig neue Klangkreationen zu
schaffen, selbst Weltraumklänge. Verwandeln Sie Ihre
64er für wenig Geld in eine Super-Musikmaschine mi
SYNTHIMAT. Komplett mit ausführlichem Handbuch
nur DM 99,-.



FAKTUMAT

Eine Sofortfakturierung mit integrierter Lagerbuch-
führung. Ideal für jeden Kleinbetrieb durch individuel
Anpassung von Steuersätzen, Maßeinheiten und Firm
daten an eigene Bedürfnisse. Natürlich sind auch die
Kunden- und die Artikelstammdatei voll pflegbar. So
können Sie beliebig den Umfang der Dateien wählen u
diese Ihren Erfordernissen anpassen. Durch eine bes
dere Programmierweise ist es möglich, sehr schnell a
Kunden- und Artikeldaten zurückzugreifen. Der Zugri
auf diese Daten erfolgt jeweils über einen 6stelligen
Schlüssel, den Sie frei definieren können. Die Fort-
schreibung von Artikel- und Kundendaten erfolgt selt
verständlich automatisch. Komplett mit ausführlichem
Handbuch nur DM 99,-.



KONTOMAT

Ein Einnahme-Überschußprogramm nach §4 (3) EStG
Kassenbuch, Bankkontenüberwachung, automatische
Steuerbuchung (Brutto und Netto), AfA Tabellen-
erstellung, Kontenblättern, Ermittlung der Ust.-Vorann
dungswerte und Monats- und Jahresabrechnung,
geschrieben von einem Buchhalter und einem
Programmierer. KONTOMAT ist voll parameterisiert
(Firmendaten, Steuersätze, Kontennamen . . .) und läi
sich damit an Ihre Bedürfnisse anpassen. KONTOMAT
geeignet für alle Gewerbetreibenden, die nicht laut H
zur Buchführung verpflichtet sind. Mehrere Finanzäm
haben die mit KONTOMAT ermittelten Daten bereits
erkannt. KONTOMAT ist menügesteuert und dokume
tiert sich weitgehend selbst. KONTOMAT ist für den
gewerblichen Einsatz aber auch als Lernprogramm o
zur Haushaltsbuchführung geeignet. Komplett mit au
führlichem Handbuch nur DM 99,-.

